DBMaker

JSONPath User's Guide

Version: 01.00

Document No: 45049-243188/DBM545-M04172023-JSON
Author: Production Team
Syscom Computer Engineering CO.

Print Date: April 17, 2023

凌羣電腦
THE SYSCOM GROUP

Table of Content

1. Introduction

Welcome to the DBMaker JSONPath User's Guide. This guide discusses the details about the JSONPath expressions and guides the user how to use them. Including how to insert, update, delete and select data with JSONPath.

1.1. Preface of JSONPath

After DBMaker version 5.4.5. Users can use the JSONPath expressions. Jsonpath expressions can be used on inserting, updating, deleting and selecting jsoncols. DBMaker also provides jsoncols and JSONPath functions.

JSONPath variables listed following:

JSONPath Variable	Description
$	A variable representing the root object/elements of the JSON.
Varname	A named variable. Its value can be set by the parameter vars of several JSON processing functions
@	A variable representing the current result of path evaluation in filter expressions.

JSONPath operators listed following:

JSONPath Operator	Description
.key .varname	Member accessor that returns an object member with the specified key/varname.
*	Wildcard member accessor
.*	Wildcard member accessor that returns the values of all members located at the root level of the current object.
..*	Recursive wildcard member accessor that processes all levels of the JSON hierarchy of the current object and returns all the member values.
[]	Array element accessor. Subscript index zero corresponds to the first array element. The specified index can be an integer, as well as an expression returning a single numeric value, which is automatically cast to integer.

[subscript, ...]	Array element accessor. This Subscript operator is array index. The form returns a single array element by its index. The specified index can be an integer, as well as an expression returning a single numeric value, which is automatically cast to integer.
[start:end:step]	Array slice accessor. This Subscript operator is including the elements start_index to end_index and step. The form returns an array slice by the range of indexes, correspond to the provided start_index and end_index. The step default means 1 for integer. You can also use the last keyword to denote the last array element.
?()	Applies a filter(script) expression
()	Script expression

2. Query JSON Data with JSONPath

This chapter tells how to query the JSON data with JSONPath. The returned value is always a VARCHAR string instance that represents a JSON value. Following are examples, detail cases and grammar expressions.

Use the following command to create table “city_stores”:

This table will be used on following examples.

```
dmSQL> CREATE TABLE city_stores
 ( id int NOT NULL PRIMARY KEY,
  tag VARCHAR(32),
  json_doc JSONCOLS);


dmSQL> INSERT INTO city_stores VALUES (1,’comment 1’, ‘
{ "store" :
 {
  "book" : [
   { "category": "reference",
    "author": "Nigel Rees",
    "title": "Sayings of the Century",
    "price": 8.95
   },
   { "category": "fiction",
    "author": "Evelyn Waugh",
    "title": "sword of Honour",
    "price": 12.99
   },
   { "category": "fiction",
    "author": "Herman Melville",
    "title": "Moby Dick",
    "isbn": "0-553-21311-3",
```

```
      "price": 8.99
    },
    { "category": "fiction",
      "author": "J. R. R Tolkien",
      "title": "The Load of the Rings",
      "isbn": "0-395-19395-8",
      "price": 22.99
   }
   ],
   "bicycle": {
     "color": "red",
     "price": 19.95
   }
 }
}');
```

2.1. Basic Query JSONPath Expressions

2.1.1. CHILD MEMBER ACCESS

One JSON object may have multiple child members, users can limit the output results by selecting child members.

➲Example 1:

This example will use the following JSONPath expression.

$.store.bicycle.price

The following query extracts from JSON column json_doc, returns the value of field “price” for the “bicycle” in JSON column “store”.

```
dmSQL> SELECT "$.store.bicycle.price" FROM city_stores;


                    $.store.bicycle.price
=========================================================================
19.95


1 rows selected
```

➲Example 2:

This example will use the following JSONPath expression.

$.store.book

The following query extracts from each JSON document, returns an array of JSON objects, which is the value of field “book” of the object that is the value of field “store”.

```
dmSQL> select "$.store.book" from city_stores;


                    $.store.book
=========================================================================
[{"title":"Sayings of the Century","author":"Nigel Rees","category":"reference"*


1 rows selected
```

2.1.2. ARRAY ELEMENT ACCESS

➲Example 1:

This example will use the following JSONPath expression.

$.store.book[0]

The following query extracts one set of arrays of JSON objects, which is the value of subscript “0” of array field “book” of the object that is the value of field “store”.

```
dmSQL> select "$.store.book[0]" from city_stores;


                    $.store.book[0]
=========================================================================
{"title":"Sayings of the Century","category":"reference","price":8.95,"author":*


1 rows selected
```

2.2. Element/Array Query JSONPath Expressions

2.2.1. ‘*’ WILDCARD EXPRESSION

Scan and find all fields in the current layer which the ‘*’ point to in the JSON document.

➲Example 1:

This example will use the following JSONPath expression.

$.*

The following query extracts from each document, multiple values as an array: any field of the JSON will be matched. The “*” make a full wildcard for the field name at first level then mark the matched value into a new array.

Note: The order of the array elements is unsupported.

```
dmSQL> select "$.* " from city_stores;


                    $.*
```

```
==========================================================================
[{"book":[{"title":"Sayings of the Century","category":"reference","price":8.95*


1 rows selected
```

➲Example 2:

This example will use the following JSONPath expression.

$.store.bicycle.*

The following query extracts from each document, multiple values as an array: any field of the JSON will be matched. The “*” make a full wildcard for the all object in the filed “store.bicycle”, then make value into a new array.

Note: The order of the array elements is unsupported.

```
dmSQL> select "$.store.bicycle.*" from city_stores;


                    $.store.bicycle.*
==========================================================================
[19.95,"red"]


1 rows selected
```

➲Example 3:

This example will use the following JSONPath expression.

$.store.book[*].author

The following query extracts from each document, multiple values as an array: the value of field “author” for each object in array “book”. The returned array is not part of the stored data but is constructed automatically by the query.

Note: The order of the array elements is unsupported.

```
dmSQL> select "$.store.book[*].author" from city_stores;


                  $.store.book[*].author
==========================================================================
["Herman Melville","Evelyn Waugh","Nigel Rees","J. R. R Tolkien"]


1 rows selected
```

➲Example 4:

This example will use the following JSONPath expression.

$.store.book[0].*

The following query extracts the values of one array set of JSON objects, which is the value of subscript “0” of array field “book” of the object that is the value of field “store”. Then “*” make the extracts value into a new array.

```
dmSQL> select "$.store.book[0].*" from city_stores;

                    $.store.book[0].*
=======================================================================
["reference","Sayings of the Century",8.95,"Nigel Rees"]

1 rows selected
```

2.2.2. '..' WILDCARD EXPRESSION

Scan and find every fields in the scope which the '..' point to in the JSON document.

The scope includes the contents of all layers owned by the current name of scope.

➲Example 1:

This example will use the following JSONPath expression.

$..author

The following query extracts the values from the full JSON objects, which is the value of filed "author" for all the JSON documents. The ".." make a full scan for the filed name then mark the matched value into a new array.

```
dmSQL> select "$..author" from city_stores;

                      $..author
=======================================================================
["Nigel Rees","Herman Melville","Evelyn Waugh","J. R. R Tolkien"]

1 rows selected
```

➲Example 2:

This example will use the following JSONPath expression.

$.store..price

The following query extracts the values from the JSON objects, which is the value of filed "price" for the field of the "store" in JSON column. The ".." make a full scan for the filed name then mark the matched value into a new array.

```
dmSQL> select "$.store..price" from city_stores;

                    $.store..price
=======================================================================
[19.95,8.95,22.99,12.99,8.99]

1 rows selected
```

➲Example 3:

This example will use the following JSONPath expression.

$..*

The following query extracts the values from the JSON objects, which is the value of all filed in JSON column. The “..” make a full scan for the filed name then put the all value into a new array.

```
dmSQL> select "$..*" from city_stores;

                        $..*
========================================================================
[22.99,"Nigel Rees",[{"title":"Sayings of the Century","price":8.95,"author":"N*

1 rows selected
```

2.2.3. ARRAY SLICE EXPRESSION

The slice operation on the array includes the slice and aggregation of array data.

➲Example 1:

This example will use the following JSONPath expression.

$..book[-1:]

The following query extracts the array values from the JSON objects, which is the value of filed “book” for the all in JSON column. ”-1” in slice means the last of the array in the order of the object in array “book”.

```
dmSQL> select "$..book[-1:]" from city_stores;

                      $..book[-1:]
========================================================================
[{"price":22.99,"title":"The Load of the Rings","isbn":"0-395-19395-8","categor*

1 rows selected
```

➲Example 2:

This example will use the following JSONPath expression.

$..book[-2:-1]

The following query extracts the array values from the JSON objects, which is the value of filed “book” for the all in JSON column.”-2:-1” in slice means the last two to last book in the order of the object in array “book”. The array has 4 book, so it’s same as “2:3”.

```
dmSQL> select "$..book[-2:-1]" from city_stores;

                      $..book[-2:-1]
```

```
==========================================================================
[{"price":8.99,"title":"Moby Dick","isbn":"0-553-21311-3","category":"fiction",*


1 rows selected
```

➲Example 3:

This example will use the following JSONPath expression.

$..book[0:3]

$..book[:3]

The following queries extract the array values from the JSON objects, which is the value of filed “title” for the all field of “book” in the JSON column. ”0:3” in slice means the first to the third of the array in the order of the object in array “book”.

```
dmSQL> select "$..book[0:3].title" from city_stores;


                    $..book[0:3].title
==========================================================================
["Sayings of the Century","sword of Honour","Moby Dick"]


1 rows selected


dmSQL> select "$..book[:3].title" from city_stores;


                    $..book[:3].title
==========================================================================
["sword of Honour","Sayings of the Century","Moby Dick"]


1 rows selected
```

➲Example 4:

This example will use the following JSONPath expression.

$.store..book[0:3:1]

$.store..book[0:3:2]

$.store..book[0:3:3]

The following queries extract the array values from the JSON objects, which return the value of “0” and “3” subscript array slice in the array in the order of field “book” for the field of the “store” in JSON column. The first two numbers determine the starting number and the ending number.

The third number determines the value of step from the starting to the ending.

```
dmSQL> select "$..book[0:3:1].title" from city_stores;
```

```
                    $..book[0:3:1].title
========================================================================
["sword of Honour","Sayings of the Century","Moby Dick"]

1 rows selected

dmSQL> select "$..book[0:3:2].title" from city_stores;

                    $..book[0:3:2].title
========================================================================
["Sayings of the Century","Moby Dick"]

1 rows selected

dmSQL> select "$..book[0:3:3].title" from city_stores;

                    $..book[0:3:3].title
========================================================================
["Sayings of the Century"]

1 rows selected
```

2.2.4. UNION FOR ARRAY EXPRESSION

➲Example 1:

This example will use the following JSONPath expression.

$..book[0,2]

The following query extracts the array values from the JSON objects, which is return the value of “0” and “2” subscript array slice in the array in the order of field “book” for the all in JSON column.

```
dmSQL> select "$..book[0,2]" from city_stores;

                      $..book[0,2]
========================================================================
[{"title":"Sayings of the Century","author":"Nigel Rees","category":"reference"*

1 rows selected
```

➲Example 2:

This example will use the following JSONPath expression.

$.store..book[0,2,3]

The following query extracts the array values from the JSON objects, which returns the value of “0”, “2” and “3” subscript array slice in the array in the order of field “book” for the all in JSON column.

```
dmSQL> select "$.store..book[0,2,3]" from city_stores;

                    $.store..book[0,2,3]
========================================================================
[{"author":"Herman Melville","isbn":"0-553-21311-3","title":"Moby Dick","catego*

1 rows selected

dmSQL> select "$.store..book[0,2,3].title" from city_stores;

                  $.store..book[0,2,3].title
========================================================================
["Moby Dick","The Load of the Rings","Sayings of the Century"]

1 rows selected
```

2.3. Filter Query JSONPath Expressions

2.3.1. FILTER WITH CURRENT ELEMENT

➲Example:

This example will use the following JSONPath expression.

$..book[?(@.isbn)]

The following query extracts the array values from the JSON objects, which returns each object in the subscript array who has element “isbn” for all of field “book” in JSON column.

```
dmSQL> select "$..book[?(@.isbn)]" from city_stores;

                    $..book[?(@.isbn)]
========================================================================
[{"author":"Herman Melville","isbn":"0-553-21311-3","title":"Moby Dick","catego*

1 rows selected
```

```
dmSQL> select "$..book[?(@.isbn)].title" from city_stores;

                $..book[?(@.isbn)].title
================================================================
["Moby Dick","The Load of the Rings"]

1 rows selected
```

2.3.2. FILTER WITH CONDITION

➲Example 1:

This example will use the following JSONPath expression.

$..book[?(@.category=’fiction’)]

The following query extracts the array values from the JSON objects, which returns each object in the subscript array what the value of element “category” is “fiction” for all of field “book” in JSON column.

```
dmSQL> select "$..book[?(@.category='fiction')]" from city_stores;

            $..book[?(@.category='fiction')]
================================================================
[{"category":"fiction","price":8.99,"isbn":"0-553-21311-3","title":"Moby Dick",*

1 rows selected

dmSQL> select "$..book[?(@.category='fiction')].category" from city_stores;

          $..book[?(@.category='fiction')].category
================================================================
["fiction","fiction","fiction"]

1 rows selected
```

➲Example 2:

This example will use the following JSONPath expression.

$..book[?(@.category!='fiction')]

The following query extracts the array values from the JSON objects, which returns each object in the subscript array what the value of element “category” is not “fiction” for all of field “book” in JSON column.

```
dmSQL> select "$..book[?(@.category!='fiction')]" from city_stores;

              $..book[?(@.category!='fiction')]
================================================================================
[{"author":"Nigel Rees","title":"Sayings of the Century","category":"reference"*

1 rows selected

dmSQL> select "$..book[?(@.category!='fiction')].category" from city_stores;

            $..book[?(@.category!='fiction')].category
================================================================================
["reference"]

1 rows selected
```

➲Example 3:

This example will use the following JSONPath expression.

$..book[?(@.price<10)], $..book[?(@.price>10)]

The following queries extract the array values from the JSON objects, which return each object in the subscript array what the value of element “price” is lower or upper then 10 for all of field “book” in JSON column.

```
dmSQL> select "$..book[?(@.price<10)]" from city_stores;

                  $..book[?(@.price<10)]
================================================================================
[{"author":"Nigel Rees","title":"Sayings of the Century","category":"reference"*

1 rows selected

dmSQL> select "$..book[?(@.price<10)].price" from city_stores;

                $..book[?(@.price<10)].price
================================================================================
[8.95,8.99]

1 rows selected
```

```
dmSQL> select "$..book[?(@.price>10)]" from city_stores;

                $..book[?(@.price>10)]
================================================================================
[{"author":"Evelyn Waugh","title":"sword of Honour","category":"fiction","price*

1 rows selected

dmSQL> select "$..book[?(@.price>10)].price" from city_stores;

                $..book[?(@.price>10)].price
================================================================================
[12.99,22.99]

1 rows selected
```

➲Example 4:

This example will use the following JSONPath expression.

$..book[?(@.price<=12.99)], $..book[?(@.price>=12.99)]

The following queries extract the array values from the JSON objects, which return each object in the subscript array what the value of element “price” is upper (lower) than or equal to 12.99 for all of field “book” in JSON column.

```
dmSQL> select "$..book[?(@.price>=12.99)].price" from city_stores;

                $..book[?(@.price>=12.99)].price
================================================================================
[12.99,22.99]

1 rows selected

dmSQL> select "$..book[?(@.price<=12.99)].price" from city_stores;

                $..book[?(@.price<=12.99)].price
================================================================================
[12.99,8.99,8.95]
```

```
1 rows selected
```

➲Example 5:

This example will use the following JSONPath expression.

$..book[?(@.price=8.95)], $..book[?(@.price==8.95)]

$..book[?(@.price<>8.95)], $..book[?(@.price!=8.95)]

The following queries extract the array values from the JSON objects, which return each object in the subscript array what the value of element “price” is equal or unequal to 8.95 for all of field “book” in JSON column.

```
dmSQL> select "$..book[?(@.price=8.95)]" from city_stores;

                    $..book[?(@.price=8.95)]
======================================================================
[{"author":"Nigel Rees","title":"Sayings of the Century","category":"reference"*

1 rows selected

dmSQL> select "$..book[?(@.price==8.95)]" from city_stores;

                    $..book[?(@.price==8.95)]
======================================================================
[{"author":"Nigel Rees","title":"Sayings of the Century","category":"reference"*

1 rows selected

dmSQL> select "$..book[?(@.price<>8.95)]" from city_stores;

                    $..book[?(@.price<>8.95)]
======================================================================
[{"category":"fiction","price":8.99,"isbn":"0-553-21311-3","title":"Moby Dick",*

1 rows selected

dmSQL> select "$..book[?(@.price!=8.95)]" from city_stores;

                    $..book[?(@.price!=8.95)]
======================================================================
```

```
[{"category":"fiction","price":8.99,"isbn":"0-553-21311-3","title":"Moby Dick",*


1 rows selected
```

➲Example 6:

This example will use the following JSONPath expression.

$.store.book[?(30=30)]

$.store.book[?(30<>30)]

$.store.book[?(30!=30)]

The following queries extract the array values from the JSON objects, which return each object in the subscript array what the constant value “30” is equal or unequal to 30 for all of field “book” in JSON column.

```
dmSQL> select "$.store.book[?(30=30)].price" from city_stores;


                $.store.book[?(30=30)].price
====================================================================
[12.99,8.95,22.99,8.99]


1 rows selected


dmSQL> select "$.store.book[?(30!=30)].price" from city_stores;


              $.store.book[?(30!=30)].price
====================================================================
                          NULL


1 rows selected
```

➲Example 7:

This example will use the following JSONPath expression.

$..book[?(@.price<30 && @.category==’fiction’)]

$..book[?(@.price<30 AND @.category==’fiction’)]

The following queries extract the array values from the JSON objects, which return each object in the subscript array that has two condition JSON column.

The first condition checks the value of current element “price”.

The second condition checks current element “category”.

```
dmSQL> select "$..book[?(@.price<30 && @.category=='fiction')]" from city_stores;


            $..book[?(@.price<30 && @.category=='fiction')]
```

```
==========================================================================
[{"category":"fiction","price":8.99,"isbn":"0-553-21311-3","title":"Moby Dick",*

1 rows selected

dmSQL> select "$..book[?(@.price<30 and @.category=='fiction')]" from city_stores;

        $..book[?(@.price<30 and @.category=='fiction')]
==========================================================================
[{"author":"Evelyn Waugh","title":"sword of Honour","category":"fiction","price*

1 rows selected
```

➲Example 8:

This example will use the following JSONPath expression.

$.store.book[?(@.price>10 && @.price<20)]

$.store.book[?(@.price>10 and @.price<20)]

The following queries extract the array values from the JSON objects, which return each object in the subscript array who has two condition JSON column.

The first condition checks the value of current element “price” greater than 10.

The second condition checks the value of current element “price” less than 10.

The two conditions are “and” relations.

```
dmSQL> select "$.store.book[?(@.price>10 && @.price<20)].price" from city_stores;

        $.store.book[?(@.price>10 && @.price<20)].price
==========================================================================
[12.99]

1 rows selected

dmSQL> select "$.store.book[?(@.price>10 and @.price<20)].price" from city_stores;

        $.store.book[?(@.price>10 and @.price<20)].price
==========================================================================
[12.99]

1 rows selected
```

➲Example 9:

This example will use the following JSONPath expression.

$.store.book[?(@.price>10 || @.price<20)]

$.store.book[?(@.price>10 OR @.price<20)]

The following queries extract the array values from the JSON objects, which return each object in the subscript array what has two condition JSON column.

The first condition checks the value of current element “price” less than 10.

The second condition checks the value of current element “price” greater than 10.

The two conditions are “or” relations.

```
dmSQL> select "$.store.book[?(@.price<10 || @.price>20)].price" from city_stores;

            $.store.book[?(@.price<10 || @.price>20)].price
================================================================================
[8.99,22.99,8.95]

1 rows selected

dmSQL> select "$.store.book[?(@.price<10 or @.price>20)].price" from city_stores;

            $.store.book[?(@.price<10 or @.price>20)].price
================================================================================
[8.99,22.99,8.95]

1 rows selected
```

2.3.3. FILTER WITH EXPRESSION SUBSCRIPT

➲Example 1:

This example will use the following JSONPath expression.

$..book[(@.length-1)]

The following query extracts the array values from the JSON objects, which returns each object in the subscript array in JSON column. The subscript supports the expression such as “length” show the length of the current level.

```
dmSQL> select "$..book[(@.length-1)]" from city_stores;

                $..book[(@.length-1)]
================================================================================
[{"author":"J. R. R Tolkien","category":"fiction","title":"The Load of the Ring*
```

```
1 rows selected
```

➲Example 2:

This example will use the following JSONPath expressions.

$.store.book[(2+1)]

$.store.book[(4-1)]

$.store.book[(1+(1+1))]

$.store.book[(1*3)]

$.store.book[(1+(1*2))]

$.store.book[(6/2)]

$.store.book[(4%3)]

The following queries extract the array values from the JSON objects, which return each object in the subscript array in JSON column. The subscript supports basic numerical expression such as “+”, “-”, “*”, “/”, “%”.

```
dmSQL> select "$.store.book[(2+1)].title" from city_stores;

                $.store.book[(2+1)].title
=========================================================================
["The Load of the Rings"]

1 rows selected

dmSQL> select "$.store.book[(4-1)].title" from city_stores;

                $.store.book[(4-1)].title
=========================================================================
["The Load of the Rings"]

1 rows selected

dmSQL> select "$.store.book[(1+(1+1))].title" from city_stores;

              $.store.book[(1+(1+1))].title
=========================================================================
["The Load of the Rings"]
```

```
1 rows selected

dmSQL> select "$.store.book[(1*3)].title" from city_stores;

                $.store.book[(1*3)].title
============================================================================
["The Load of the Rings"]

1 rows selected

dmSQL> select "$.store.book[(1+(1*2))].title" from city_stores;

              $.store.book[(1+(1*2))].title
============================================================================
["The Load of the Rings"]

1 rows selected

dmSQL> select "$.store.book[(6/2)].title" from city_stores;

                $.store.book[(6/2)].title
============================================================================
["The Load of the Rings"]

1 rows selected

dmSQL> select "$.store.book[(7%4)].title" from city_stores;

                $.store.book[(7%4)].title
============================================================================
["The Load of the Rings"]

1 rows selected
```

3. Insert/ Update/ Delete JSON data with JSONPath

All of the usual database function used to insert/update the dynamic column can also be used on JSON columns. The column of data type JSON is always well-formed JSON data.

If user inserts a JSON column through the client, the SQL will make a constraint check for the column to make sure the inserted JSON data is well-formed JSON data.

If user updates a JSON column through the client, the updated dynamic column must be a well-formed JSON data, otherwise the SQL command will occur an error.

The database handles the dynamic column as a domain, and the JSON check constraint is the same usual domain constraint.

The dynamic column works with a column of type BLOB and inserts with a type CLOB that contains JSON string. There is no difference from working with any other column of these type.

Inserting a JSON string into a dynamic column or updating the data in such a column is convenience if the column is jsoncol domain.

3.1. Insert JSON Data

DBMaker can insert into the JSON column which supports a whole JSON format. Unlike relational data, JSON data can be stored in tables and indexes, it is queried without any need for a schema that defines the data.

The following example shows how to insert JSON data:

```
dmSQL> create table test(c0 jsoncols);


dmSQL> insert into test(c0) values('
{ "store" :
 {
   "book" : [
    { "category": "reference",
      "author": "Nigel Rees",
      "title": "Sayings of the Century",
      "price": 8.95
```

```
    },
    { "category": "fiction",
      "author": "Evelyn Waugh",
      "title": "sword of Honour",
      "price": 12.99
    },
    { "category": "fiction",
      "author": "Herman Melville",
      "title": "Moby Dick",
      "isbn": "0-553-21311-3",
      "price": 8.99
    },
    { "category": "fiction",
      "author": "J. R. R Tolkien",
      "title": "The Load of the Rings",
      "isbn": "0-395-19395-8",
      "price": 22.99
   }
   ],
   "bicycle": {
     "color": "red",
     "price": 19.95
   }
 }
}');
1 rows inserted
```

The update/delete chapters below will base on this example.

3.2. Update JSON Data with JSONPath

DBMaker makes DML operation in the dynamic column by using JSONPath with the update statements. The dynamic column which is updated in the table must be an existing column. When the table doesn't have dynamic columns or the record which is updated in the dynamic column doesn't exist, the updating does not any change.

3.2.1. UPDATE EXIST COLUMN WITH JSONPATH

Usually, the update operation will only update the existing columns, but for dynamic columns, updating the non-existing columns can also update the data.

3.2.2. UPDATE SINGLE LAYER WITH JSONPATH

➲Example 1:

The following example shows the field “store” of the document is updated, include all the object in this field.

```
dmSQL> select "$.store" from city_stores;

                        $.store
========================================================================
{"bicycle":{"price":19.95,"color":"red"},"book":[{"title":"Sayings of the Centu*

1 rows selected

dmSQL> update city_stores set "$.store" = 'replace value';
1 rows updated

dmSQL> select json_doc from city_stores;

                        JSON_DOC
========================================================================
{"store":"replace value"}

1 rows selected
```

3.2.3. UPDATE MULTI-LAYER WITH JSONPATH

➲Example 1:

The following example shows the field “$.store.book[0].title” of the document is updated, include all the object in this field.

```
dmSQL> select "$.store.book[1].author" from city_stores;

                    $.store.book[1].author
========================================================================
Evelyn Waugh
```

```
1 rows selected

dmSQL> update city_stores set "$.store.book[1].author" = 'updating Evelyn Waugh';
1 rows updated

dmSQL> select "$.store.book[1].author" from city_stores;

                $.store.book[1].author
=======================================================================
updating Evelyn Waugh

1 rows selected
```

➲Example 2:

The following example shows how to update the array and object with JSONPath.

```
dmSQL> select "$.store.book[0]" from city_stores;

                  $.store.book[0]
=======================================================================
{"author":"Nigel Rees","category":"reference","price":8.95,"title":"Sayings of *

1 rows selected

dmSQL> update city_stores set "$.store.book[0]" = 'null book' ;
1 rows updated

dmSQL> select "$.store.book[0]" from city_stores;

                  $.store.book[0]
=======================================================================
null book

1 rows selected
```

3.2.4. UPDATE WITH JSONPATH EXPRESSION

When updating with JSONPath, users must ensure the JSONPath of updated object is correct, the wrong operation may break the hierarchical object.

The updating also supports the query expression such as wildcard “*”, “..”, slice, union expression and filter expression.

3.2.4.1. Update with JSONPath and “*” Wildcard Expression

➲Example:

The following example shows all query result include wildcard “*” will be updated.

```
dmSQL> select "$.store.bicycle.*" from city_stores;

                    $.store.bicycle.*
=======================================================================
["red",19.95]

1 rows selected

dmSQL> update city_stores set "$.store.bicycle.*" = 'updating value';
1 rows updated

dmSQL> select "$.store.bicycle.*" from city_stores;

                    $.store.bicycle.*
=======================================================================
["updating value","updating value"]

1 rows selected
```

3.2.4.2. Update with JSONPath and “..” Wildcard Expression

➲Example:

The following example shows all query result include wildcard “..” will be updated.

```
dmSQL> select "$.store..title" from city_stores;

                      $.store..title
=======================================================================
["The Load of the Rings","Moby Dick","sword of Honour"]

1 rows selected

dmSQL> update city_stores set "$.store..title" = 'updating ..';
1 rows updated
```

```
dmSQL> select "$.store..title" from city_stores;

                    $.store..title
=======================================================================
["updating ..","updating ..","updating .."]

1 rows selected
```

3.2.4.3. Update with JSONPath and Slice and Union Expression

➲Example:

The following example shows the all query result include slice expression will be updated.

```
dmSQL> select "$.store.book[*].title" from city_stores;

                  $.store.book[*].title
=======================================================================
["Sayings of the Century","sword of Honour","The Load of the Rings","Moby Dick"]

1 rows selected

dmSQL> update city_stores set "$.store.book[0:2:1].title" = 'updating title';
1 rows updated

dmSQL> select "$.store.book[*].title" from city_stores;

                  $.store.book[*].title
=======================================================================
["The Load of the Rings","updating title","updating title","Moby Dick"]

1 rows selected
```

3.2.4.4. Update with JSONPath and Filter Expression

➲Example:

The following example shows all the query result include filter value will be updated.

```
dmSQL> select "$..book[*].price" from city_stores;

                    $..book[*].price
```

```
==========================================================================
[22.99,12.99,8.95,8.99]

1 rows selected

dmSQL> update city_stores set "$..book[?(@.price>8.95)].price" = '0';
1 rows updated

dmSQL> select "$..book[*].price" from city_stores;

                    $..book[*].price
==========================================================================
[8.95,"0","0","0"]

1 rows selected
```

3.2.5. UPDATE NONE EXIST DATA WITH JSONPATH

If the data to update does not exist, then DBMaker will insert the data into the JSON column of the table.

➲Example 1:

The following example shows if user updates a none exist data.

```
dmSQL> create table test(c0 jsoncols);

dmSQL> insert into test values();
1 rows inserted

dmSQL> select * from test;

                         C0
==========================================================================
                       NULL

1 rows selected

dmSQL> update test set "$.C1"='upd_value';
1 rows updated
```

```
dmSQL> select * from test;

                    C0
==========================================================================
{"C1":"upd_value"}

1 rows selected
```

➲Example 2:

The following example shows the parent objects can also be created when updating the none exist data.

```
dmSQL> select * from test;

                    C0
==========================================================================
{"C1":"upd_value"}

1 rows selected

dmSQL> update test set "$.C2.C1"='upd_value';
1 rows updated

dmSQL> select * from test;

                    C0
==========================================================================
{"C2":{"C1":"upd_value"},"C1":"upd_value"}

1 rows selected
```

➲Example 3:

The following example shows the update statement with JSONPath "$.A.B.C[0].D" will create the object of each level in the JSONPath to make sure the updating success.

```
dmSQL> select * from test;
```

```
                    C0
=====================================================================
                    NULL

1 rows selected

dmSQL> update test set "$.A.B.C[0].D"='value';
1 rows updated

dmSQL> select * from test;

                    C0
=====================================================================
{"A":{"B":{"C":[{"D":"value"}]}}}

1 rows selected
```

➲Example 4:

The following example shows the update statement with JSONPath whose subscript of array object is 1, when updating whose array subscript is not 0, DBMaker will fill the array with null to make the operation success.

```
dmSQL> select * from test;

                    C0
=====================================================================
                    NULL

1 rows selected

dmSQL> update test set "$.C1[1]"='array_value _1';
1 rows updated

dmSQL> select * from test;

                    C0
=====================================================================
```

```
{"C1":[null,"array_value _1"]}


1 rows selected
```

3.3. Delete JSON Data with JSONPath

The delete operation is divided into deleting the record of dynamic column or deleting the field and sub subscript in the dynamic column.

3.3.1. DELETE JSON COLUMN

When users need to delete the entire column of data or delete the entire JSON documents, use SQL command "delete".

The same as the normal delete command, the deleting remove the columns including JSON column.

➲Example:

The following command can delete the whole JSON document.

```
dmSQL> delete from table_name;
```

3.3.2. DELETE JSON OBJECT WITH JSONPATH

When it is necessary to delete some field of the data in the entire JSON document, use the SQL command "update" then set the deleted field to NULL.

The same as the normal delete command, the update statement remove the columns including JSON column.

Users can update dynamic column with column name, using JSONPath can get the same effect.

➲Example 1:

The following example shows how to delete the selected data with update statement.

```
dmSQL> select * from test;


                        C0

==========================================================================

{"C1":"record1","C2":"record2","C3":"record3"}


1 rows selected


dmSQL> update test set C1= NULL;

1 rows updated
```

```
dmSQL> select * from test;

                    C0
========================================================================
{"C2":"record2","C3":"record3"}
```

➲Example 2:

The following example shows how to delete the selected data by JSONPath with update statement.

```
dmSQL> select * from test;

                    C0
========================================================================
{"C2":"record2","C3":"record3"}

dmSQL> update test set "$.C2"= NULL;
1 rows updated

dmSQL> select * from test;

                    C0
========================================================================
{"C3":"record3"}

1 rows selected
```