

Objects JSON into fields of tables



Disclaimer

This SOFTWARE PRODUCT is provided by El Condor "as is" and "with all faults." El Condor makes no representations or warranties of any kind concerning the safety, suitability, lack of viruses, inaccuracies, typographical errors, or other harmful components of this SOFTWARE PRODUCT. There are inherent dangers in the use of any software, and you are solely responsible for determining whether this SOFTWARE PRODUCT is compatible with your equipment and other software installed on your equipment. You are also solely responsible for the protection of your equipment and backup of your data, and El Condor will not be liable for any damages you may suffer in connection with using, modifying, or distributing this SOFTWARE PRODUCT.

You can use this SOFTWARE PRODUCT freely, if you would you can credit me in program comment:

El Condor – CONDOR INFORMATIQUE – Turin

Comments, suggestions and criticisms are welcomed: mail to rossati@libero.it

Conventions

Commands syntax, instructions in programming language and examples are with font **COURIER NEW**.
The optional parties of syntactic explanation are contained between [square parentheses],
alternatives are separated by | and the variable parties are in *italics*.

Contents

1 Reasons for the project.....	1
2 JSON object into Data Base tables.....	1
2.1 SQLite.....	1
2.2 MySql and MariaDB.....	1
3 The tool.....	1
3.1 Table generation.....	1
3.1.1 Function and method.....	1
3.1.2 Data extracted.....	2
3.1.3 Data from SQL SELECT.....	2
3.1.4 Data from array of JSON fields.....	3
3.1.5 Dealing with equal name fields.....	4
3.1.6 Create a permanent table.....	5
3.2 Execute SQL.....	5
3.3 Extract fields.....	5
3.3.1 Extract fields from string.....	5
3.3.2 Extract fields from array.....	6
4 Errors.....	6
5 Compatibility.....	6
6 History.....	6

1 Reasons for the project

In an application there is often a need to store information which, despite being classifiable under a common typology, nonetheless contain different data, for example a table of events that contains the information like logging, particular activities on data and so on; a minimal solution can be a table with three fields:

- event type,
- JSON data object,
- timestamp (if not in the above field).

Where the JSON object contains data that depends on the event type.

However, with this solution, the problem arises when we want to recover the data of a certain type to process it in a form compatible with relational tools id est a table, so this work shows a solution to this problem using the PHP Data Objects extension (PDO).

2 JSON object into Data Base tables

2.1 SQLite

The JSON functions and operators were introduced in the SQLite release 3.37.2 and are built by default from the version 3.38.0 (2022-02-22)¹.

The PHP Class described in this document does not use the features offered by SQLite and can therefore be used with versions of SQLite lower than those mentioned above.

2.2 MySql and MariaDB

The package has been tested with MariaDB and therefore is Mysql compatible.

3 The tool

The script `json2table.php` contains the class `J2t` that has the method:

- `json2t`

and three static public functions:

- `json2table`
- `execSQL`
- `extractFields`

The method and the first two static functions assume that the database has already been opened (i.e. by creation of a new PDO object).

The scope of the method and the first one function is to create a temporary table using an array of Jason Objects or an SQL statement, the second function, that is also used internally, executes SQL statements; the third function returns data extracted from a string or array, presumably a row of data with any JSON fields.

3.1 Table generation

3.1.1 Function and method


The `json2table` function and `json2t` method structure are:

```
json2t[able]($dbh,$data,$tableName="TemporaryTable",$flds = array())
```

where:

- `$dbh` is an instances of a PDO object,

¹ See <https://www.sqlite.org/json1.html#jptr>

- `$data` is an array of JSON objects or an SQL SELECT statement concerning table(s) that can contain JSON data,
- `$tableName` is the name of the temporary table created,  the table isn't generate if it exists in the database,
- `$flds` is an optional array or a string of field names to extract from JSON fields.

`json2t[able]` returns an empty string if it succeed otherwise return an error message (see par. 4 Errors).

The function can be called:

```
J2t::json2table
```

```
ex. echo J2t::json2table($db,$sql,$tTable,$fields);
```

the method:

```
$obj->j2t;
```

```
ex. $j2table = new J2t;
```

```
$j2table->j2t($db,$sql,$tTable,$fields);
```

3.1.2 Data extracted

The first set of data determines the content of the output table, that is, this will contain all the non JSON fields and all the fields extracted from the JSON fields, however, for the latter, it is possible to indicate in the fourth parameter only the fields desired and possibly fields not contained in the first data set.

The usefulness of the fourth parameter is in being able:

- to obtain a subset of the data present,
- to possibly indicate the type of data, for example, treating character data as numeric,
- to indicate field(s) not present in the first data set or row.

The field data type is inferred from the input data unless indicated in the fourth parameter; the value accepted are TEXT, INTEGER and REAL.

First data

```
{"Store":"Georgia","ToStore":"Butea","Qty":"711","Items":0.5,"Lot":"12","Box":"2758,2761,2764,2767,2770,2771,2779"}
```

Fields to extract

```
array("Store","Qty"=>"INTEGER","Items","Box","noField")
```

CREATE TABLE command

```
CREATE TEMPORARY TABLE tTable (Store TEXT,Qty INTEGER,Items REAL,Box TEXT,noField TEXT)
```

3.1.3 Data from SQL SELECT

The second parameter of the `json2table` method can be a SQL SELECT command where one or more fields can contain a JSON object.

The JSON field is an *object* i.e. a set of name/value pairs, see below.

```
{"Author":"Alan Perlis","Source":"","Quote":"Syntactic sugar causes cancer of the semicolon."}
```

```
[{"_id":"9sgo4-hjyyq","author":"The Buddha","content":"Radiate boundless love towards the entire world – above, below, and across – unhindered, without ill will, without enmity.","tags":["Wisdom","Love"],"authorSlug":"the-
```

```
buddha","length":122,"dateAdded":"2023-03-30","dateModified":"2023-04-14"]}]
```

A value can be an array (see above the key `tags`), whose elements are transformed into comma-separated strings.

Note that the SQL command can also contain non JSON fields which will be included along with the JSON generated fields; for possibly field name collision see parag. 3.1.5 *Dealing with equal name fields*.

Example 1: JSON from SELECT statement

```
$tTable = "tTable";
$table = ($Table == "Q") ? "Quotes" : "ITQuotes";
...
J2t::json2table($db,"SELECT Count, Quote, SUBSTR(Timestamp,1,10) Date FROM
$table",$tTable);
...
Array
(
    [0] => Array
        (
            [Count] => 0
            [Quote] => {"Author":"Alan Perlis","Source":"","Quote":"Syntactic
sugar causes cancer of the semicolon."}
            [Date] => 2023-05-09
        )
    [1] => Array
        (
            [Count] => 1
            [Quote] => {"Author":"Edsger Dijkstra","Source":"","Quote":"Object-
oriented programming is an exceptionally bad idea which could only have originated
in California."}
            [Date] => 2023-05-09
        )
    ...
)
```

3.1.4 Data from array of JSON fields

The second parameter of the `json2t[able]` function can be a simple array such as those obtained by fetching a single column or an array where the items are array containing associative data².

Example 2: JSON fields array

```
$tTable = "tTable";
$table = ($Table == "Q") ? "Quotes" : "ITQuotes";
...
$sql = "SELECT Quote FROM $table";
$sth = $db->prepare($sql);
$sth->execute();
$result = $sth->fetchAll(PDO::FETCH_COLUMN,0);
J2t::json2table($db,$result,$tTable);
...
Array
(
    [0] => {"Author":"Alan Perlis","Source":"","Quote":"Syntactic sugar causes
cancer of the semicolon."}
)
```

² When the function is invoked with SQL SELECT, data are in this form.

```
[1] => {"Author":"Edsger Dijkstra","Source":"","Quote":"Object-oriented programming is an exceptionally bad idea which could only have originated in California."}
[2] => {"Author":"Edsger Dijkstra","Source":"","Quote":"It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration."}
[3] => {"Author":"Edsger Dijkstra","Source":"","Quote":"When FORTRAN has been called an infantile disorder, full PL\1, with its growth characteristics of a dangerous tumor, could turn out to be a fatal disease."}
...
```

Example 3: Use whit array of data and call the function

```
...
include 'common/json2table.php';
$dbh = new PDO("sqlite:DataBaseFile");
...
$sql = "SELECT Data FROM Loggin WHERE Type = 'MoveDrugs'";
$stmt = $dbh->prepare($sql);
$stmt->execute();
$result = $stmt->fetchAll(PDO::FETCH_COLUMN, 0);
$table = "tTable";
J2t::json2table($dbh,$result,$table);
...
```

Example 4: Use whit SQL and call of class method

```
...
include 'common/json2table.php';
$db = new PDO("sqlite:DataBaseFile");
...
$sql = "SELECT Data FROM Loggin WHERE Type = 'MoveDrugs'";
$y2t = new J2t;
$answer = $y2t->json2t($db,$sql,"Temp");
if ($answer != "") exit($answer); // Something went wrong
$result = J2T::execSql($db,"SELECT Count(*) Count, SUM(Qty) Qty FROM Temp");
if (gettype($result) == "string") echo $result; // SQL problem
else print_r($result->fetchAll(PDO::FETCH_ASSOC));
$sql = "SELECT * FROM Temp WHERE Qty > ?";
print_r(J2T::execSql($db,$sql,Array(500))->fetchAll(PDO::FETCH_ASSOC));...
```

3.1.5 Dealing with equal name fields

If a field name also appears in JSON object(s), the name assigned to the latter will have the suffix `_` followed by the name of the JSON field.

Suppose the row obtained by `SELECT Date, Data, Data2 FROM Advanced` contains:

Date	2024-02-17 08:07:34
Data	{"Store":"Georgia","ToStore":"Butea","Qty":"711","Items":0.5,"Lot":"12","Box":"2758,2761"}
Data2	{"Store":"Piscina","AtStore":"Georgia","Date":"March 17","Qty":21,"Items":5.0,"Lot":"19","Box":"16"}

And the requested fields of the example are (the fourth parameter):

```
Array("Date", "Store", "AtStore", "Box" => "TEXT", "Qty" => "INTEGER")
```

The fields of the temporary table are:

```
Date TEXT, Store TEXT, AtStore TEXT, Box TEXT, Qty INTEGER, Date_Data2 TEXT,  
Store_Data2 TEXT, Box_Data2 TEXT, Qty_Data2 INTEGER
```

3.1.6 Create a permanent table

Permanent tables can be generated by setting the property `newTable = true`.

```
$j2table = new J2t;  
$j2table->newTable = true;  
$sql = "SELECT Date, Data FROM Advanced";  
echo $j2table->json2t($dbh,$sql,$tTable,"Date, ToStore, Store, Qty");
```

3.2 Execute SQL

The function `execSQL` is used by the `json2table` function and can be used for execute SQL commands:

```
execSQL($dbh,$sql,$parms=[])
```

The optional `$parms` is an array of values with the elements that are bound in the SQL statement being executed, see the *Example 4: Use whit SQL and call of class method* above.

The function returns a `PDOStatement` object or an error if the SQL isn't correct, in order to obtain the error the function sets the error mode to `SILENT`, the previous mode is restored after the command execution.

Example 5: ExecSQL function

```
public static function execSQL($dbh,$sql,$parms=[]) {  
    $errMode = $dbh->getAttribute(PDO::ATTR_ERRMODE); // save previous  
ATTR_ERRMODE  
    $dbh->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_SILENT); // disable crash  
    $sth = $dbh->prepare($sql);  
    $a = $dbh->errorInfo();  
    $dbh->setAttribute(PDO::ATTR_ERRMODE,$errMode); // restore previous  
ATTR_ERRMODE  
    if (!$sth) {return "\n<br>$sql\n<br>".$a[2];}  
    $sth->execute($parms);  
    return $sth;  
}
```

3.3 Extract fields

The function `extractFields` returns an array of fields values extracted from string that can be JSON objects.

3.3.1 Extract fields from string

The string must be a valid JSON object containing key value items; the function drops the possible surrounding square brackets.

In case of error an empty array is returned.

```
print_r(J2t::extractFields('["Store":"Georgia","ToStore":"Butea","Qty":711,"Items":0.5,"Lot":"12","Box":"2758,2761"]'));  
Array  
(  
    [Store] => Georgia  
    [ToStore] => Butea  
    [Qty] => 711
```

```

[Items] => 0.5
[Lot] => 12
[Box] => 2758,2761
)

```

3.3.2 Extract fields from array

The input must have the format of the rows returned by the PDO function `fetchAll` with parameter `PDO::FETCH_ASSOC`.

If an item is an JSON object the key values contained are extracted.

4 Errors

No data!

Table `table` exists If the output table exists (SQLite, Mysql and Maria DB)

Others error like errors on JSON data, are inserted in a temporary table with name `table_Errors`.

Undefined field "`fieldName`" The requested field `fieldName` is not present in the first row of data

5 Compatibility

Tried with:

PHP	SQLite	MariaDB
5.6.40	3.8.10.2	
8.1.4	3.36.0	11.3.2

6 History

March 2023	First version
0.2.0 8 May 2023	<ul style="list-style-type: none"> Input data from SQL SELECT can contain normal fields and one or more JSON field(s). an item of JSON field can be an array.
0.3.0 March 2024	<ul style="list-style-type: none"> Choice on fields, enhanced type management, enhanced error management.
0.4.0 June 2025	<ul style="list-style-type: none"> The parameter to indicate the fields to be extracted can be a string of fields name separated by comma, the non JSON fields are always returned, can be generated a permanent table.