```
    ->   JSON_TABLE(
    ->     '[{"a":3,"b":"0"},{"a":"3","b":"1"},{"a":2,"b":1},{"a":0},{"b":[1,2]}]',
    ->     "$[*]" COLUMNS(
    ->       rowid FOR ORDINALITY,
    ->
    ->       xa INT EXISTS PATH "$.a",
    ->       xb INT EXISTS PATH "$.b",
    ->
    ->       sa VARCHAR(100) PATH "$.a",
    ->       sb VARCHAR(100) PATH "$.b",
    ->
    ->       ja JSON PATH "$.a",
    ->       jb JSON PATH "$.b"
    ->     )
    ->   ) AS  jt1;
+-------+------+------+------+------+------+--------+
| rowid | xa   | xb   | sa   | sb   | ja   | jb     |
+-------+------+------+------+------+------+--------+
|     1 |    1 |    1 | 3    | 0    | 3    | "0"    |
|     2 |    1 |    1 | 3    | 1    | "3"  | "1"    |
|     3 |    1 |    1 | 2    | 1    | 2    | 1      |
|     4 |    1 |    0 | 0    | NULL | 0    | NULL   |
|     5 |    0 |    1 | NULL | NULL | NULL | [1, 2] |
+-------+------+------+------+------+------+--------+
```

The JSON source expression can be any expression that yields a valid JSON document, including a JSON literal, a table column, or a function call that returns JSON such as `JSON_EXTRACT(t1, data, '$.post.comments')`. For more information, see Section 12.18.6, “JSON Table Functions”.

- **Data type support.** MySQL now supports use of expressions as default values in data type specifications. This includes the use of expressions as default values for the `BLOB`, `TEXT`, `GEOMETRY`, and `JSON` data types, which previously could not be assigned default values at all. For details, see Section 11.6, “Data Type Default Values”.

- **Optimizer.** These optimizer enhancements were added:

  - MySQL now supports invisible indexes. An invisible index is not used by the optimizer at all, but is otherwise maintained normally. Indexes are visible by default. Invisible indexes make it possible to test the effect of removing an index on query performance, without making a destructive change that must be undone should the index turn out to be required. See Section 8.3.12, “Invisible Indexes”.

  - MySQL now supports descending indexes: `DESC` in an index definition is no longer ignored but causes storage of key values in descending order. Previously, indexes could be scanned in reverse order but at a performance penalty. A descending index can be scanned in forward order, which is more efficient. Descending indexes also make it possible for the optimizer to use multiple-column indexes when the most efficient scan order mixes ascending order for some columns and descending order for others. See Section 8.3.13, “Descending Indexes”.

  - MySQL now supports creation of functional index key parts that index expression values rather than column values. Functional key parts enable indexing of values that cannot be indexed otherwise, such as `JSON` values. For details, see Section 13.1.15, “CREATE INDEX Statement”.

  - In MySQL 8.0.14 and later, trivial `WHERE` conditions arising from constant literal expressions are removed during preparation, rather than later on during optimization. Removal of the condition earlier