A Quick Intro to JSON Duality Views

MySQL Belgian Days 2026 - Pep Pla

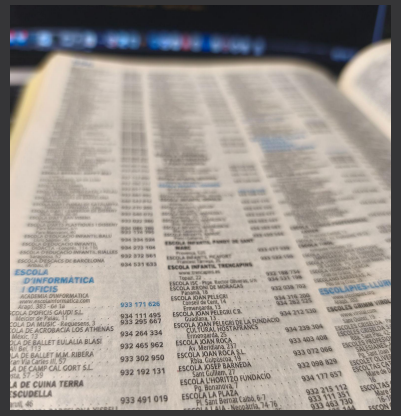
What are JSON Duality Views?

What are JSON Duality Views?

- SQL
  - Structure perfectly defined
  - Tables and columns
- NoSQL
  - Flexible structure
  - Documents and fields

What are JSON Duality Views?

- SQL
  - Structure perfectly defined
  - Tables and columns
- NoSQL
  - Flexible structure
  - Documents and fields



What are JSON Duality Views

- Data is always structured, otherwise it is not data.
- Structure may be quite simple or extremely complex.
- But JSON is an extremely convenient format for data.

What are JSON Duality Views?

- Stored queries: Views.
- Return JSON.
- Purpose: Unify structured and semi-structured data in a single view.

The Duality

- Relational side
  - Referential integrity
  - normalization
  - ACID compliance
- JSON side
  - Flexible documents
  - REST API compatibility
  - developer-friendly

MySQL 9.4+

Commu ity E D CT

M E rpr tion: Full DML

(INSERT, UPDATE, DELETE)

MySQL HeatWave

Creating JSON Duality Views

```
CREATE TABLE users (
  user_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL,
  email VARCHAR(100),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE products (
  product_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  price DECIMAL(10,2) NOT NULL,
  category VARCHAR(50)
);

CREATE TABLE shopping_carts (
  cart_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  status ENUM('active', 'checkout', 'completed', 'abandoned') DEFAULT 'active',
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(user_id)
);

CREATE TABLE cart_items (
  item_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  cart_id INT NOT NULL,
  product_id INT NOT NULL,
  quantity INT NOT NULL DEFAULT 1,
  added_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (cart_id) REFERENCES shopping_carts(cart_id),
  FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

Creating JSON Duality Views

```
CREATE JSON DUALITY VIEW jdv_users AS
SELECT JSON_DUALITY_OBJECT(
  '_id': u.user_id,
  'username': u.username,
  'email': u.email,
  'member_since': u.created_at
)
FROM users u;
```

Creating JSON Duality Views

```
{
 "_id": 1,
 "username": "alice",
 "email": "alice@example.com",
 "member_since": "2025-01-15 10:30:00"
}
```

JSON Duality Views Restrictions

```
CREATE JSON DUALITY VIEW jdv_users AS
SELECT JSON_DUALITY_OBJECT(
  '_id': u.user_id,
  'username': u.username,
  'email': u.email,
  'member_since': u.created_at
)
FROM users u;
```

- Only on base tables.
- No functions/operators.
- No joins.
- _id is required and must map to the primary key of the root object.
- Unsupported:
  - WHERE, JOIN, GROUP BY, ORDER BY
  - HAVING, WINDOW, LIMIT
  - UNION, INTERSECT, EXCEPT
  - CTEs (WITH clause)

WHAT?

Unsupported:

- WHERE, JOIN, GROUP BY, ORDER BY
- HAVING, WINDOW, LIMIT
- UNION, INTERSECT, EXCEPT
- CTEs (WITH clause)

Advanced JDV

```
CREATE JSON DUALITY VIEW shopping_cart_dv AS
SELECT JSON_DUALITY_OBJECT(
  '_id': sc.cart_id, 'status': sc.status, 'created_at': sc.created_at, 'updated_at': sc.updated_at,
  'user': (
    SELECT JSON_DUALITY_OBJECT('user_id': u.user_id, 'username': u.username, 'email': u.email)
    FROM users u
    WHERE u.user_id = sc.user_id
  ),
  'items': (
    SELECT JSON_ARRAYAGG(
      JSON_DUALITY_OBJECT('item_id': ci.item_id, 'quantity': ci.quantity, 'added_at': ci.added_at,
        'product': (
          SELECT JSON_DUALITY_OBJECT('product_id': p.product_id,'name': p.name,'price': p.price,'category': p.category)
          FROM products p
          WHERE p.product_id = ci.product_id
        )
      )
    )
    FROM cart_items ci
    WHERE ci.cart_id = sc.cart_id
  )
)
FROM shopping_carts sc;
```

Advanced JDV


```
{
  "_id": 1,
  "status": "active",
  "created_at": "2025-01-20 14:30:00",
  "updated_at": "2025-01-20 15:45:00",
  "user": {
    "user_id": 42,
    "username": "alice",
    "email": "alice@example.com"
  },
  "items": [
    {
      "item_id": 101,
      "quantity": 2,
      "added_at": "2025-01-20 14:35:00",
      "product": {
        "product_id": 5,
        "name": "Wireless Mouse",
        "price": 29.99,
        "category": "Electronics"
      }
    },
    {
      "item_id": 102,
      "quantity": 1,
      "added_at": "2025-01-20 15:45:00",
      "product": {
        "product_id": 12,
        "name": "USB-C Hub",
        "price": 49.99,
        "category": "Electronics"
      }
    }
  ]
}
```

Querying JDV

BASIC SELECT

```
SELECT * FROM
shopping_cart_dv;
```

```
SELECT
  JSON_PRETTY(data)
FROM
  shopping_cart_dv;
```

```
{
  "_id": 1,
  "status": "active",
  "user": { "user_id": 42,
"username": "alice" },
  "items": [...],
  "_metadata": {
    "etag":
"e6d40eabf2e070ffd2719c6755d50f1a"
  }
}
```

Querying JDV

Filtering with JSON Functions

```
-- Find specific cart by _id
SELECT * FROM shopping_cart_dv
WHERE JSON_VALUE(data, '$._id') = 1;

-- Find active carts
SELECT * FROM shopping_cart_dv
WHERE JSON_VALUE(data, '$.status') = 'active';

-- Find carts for specific user
SELECT * FROM shopping_cart_dv
WHERE JSON_VALUE(data, '$.user.username') =
'alice';

-- Extract cart totals (items array)
SELECT
    JSON_VALUE(data, '$._id') AS cart_id,
    JSON_LENGTH(data, '$.items') AS item_count
FROM shopping_cart_dv;
```

NULL handling in arrays

```
JSON_ARRAYAGG(
    JSON_DUALITY_OBJECT(...) NULL ON NULL    -- Include null values (default)
)

JSON_ARRAYAGG(
    JSON_DUALITY_OBJECT(...) ABSENT ON NULL  -- Omit null values
)

-- Extract cart totals (items array)
SELECT
    JSON_VALUE(data, '$._id') AS cart_id,
    JSON_LENGTH(data, '$.items') AS item_count <- NULL ITEMS COUNT!!!!!!!
FROM shopping_cart_dv;
```

Querying JDV

Filtering by Nested Array Values

```
-- Find carts containing a specific product
SELECT * FROM shopping_cart_dv
WHERE JSON_CONTAINS(
    data,
    '{"product": {"product_id": 5}}',
    '$.items'
);

-- Find carts with items in Electronics category
SELECT * FROM shopping_cart_dv
WHERE JSON_SEARCH(data, 'one', 'Electronics',
NULL, '$.items[*].product.category') IS NOT NULL;
```

DML on JDV (Enterprise Edition… well no more)

```
CREATE JSON DUALITY VIEW shopping_cart_dml_dv AS
SELECT JSON_DUALITY_OBJECT( WITH (INSERT, UPDATE, DELETE)
    '_id': sc.cart_id,
    'status': sc.status,
    'user': (
        SELECT JSON_DUALITY_OBJECT( WITH (INSERT, UPDATE)
            'user_id': u.user_id,
            'username': u.username,
            'email': u.email
        )
        FROM users u
        WHERE u.user_id = sc.user_id
    ),
    'items': (
        SELECT JSON_ARRAYAGG(
            JSON_DUALITY_OBJECT( WITH (INSERT, UPDATE, DELETE)
                'item_id': ci.item_id,
                'quantity': ci.quantity,
                'product': (
                    SELECT JSON_DUALITY_OBJECT( WITH (UPDATE)
                        'product_id': p.product_id,
                        'name': p.name,
                        'price': p.price,
                        'category': p.category
                    )
                    FROM products p
                    WHERE p.product_id = ci.product_id
                )
            )
        )
        FROM cart_items ci
        WHERE ci.cart_id = sc.cart_id
    )
)
FROM shopping_carts sc;
```

- Annotations to indicate acceptable operations.
- WARNING: wrong annotations have side effects: Modify the underlying objects.
- You can create USERS or modify PRODUCTS.

Lockless Optimistic Concurrency Control

What is this?

```
"_metadata": {
  "etag": "d66f5224abe75b0c0fd30456ead14f57"
},
```

Lockless Optimistic Concurrency Control

- The etag is the JSON document unique identifier.
- You can't update a JSON document if you provide a different etag (but you can update without providing one!)
- It is used to validate that the document read previously is still valid.

DML

```
mysql> UPDATE shopping_cart_dml_dv
    -> SET data = '{"_id": 12, "user": {"email": "r.henderson2278@yahoo.com",
"user_id": 42, "username": "raymond_henderson2278"}, "items": [{"item_id": 31,
"product": {"name": "Designing Data-Intensive Applications by Martin
Kleppmann", "price": 54.99, "category": "Books", "product_id": 87},
"quantity": 1}, {"item_id": 32, "product": {"name": "High Performance MySQL by
Baron Schwartz", "price": 59.99, "category": "Books", "product_id": 90},
"quantity": 1}, {"item_id": 33, "product": {"name": "Database Internals by
Alex Petrov", "price": 54.99, "category": "Books", "product_id": 91},
"quantity": 1}], "status": "active", "_metadata": {"etag":
"4ac9dee9b8415aeafb4b151deb6bc46a"}}'
    -> where data->'$._id' = 12;
Query OK, 3 rows affected (0.016 sec)
Rows affected: 3  Warnings: 0.
```

DML

```
mysql> UPDATE shopping_cart_dml_dv SET data = '{"_id": 12, "user": {"email":
"r.henderson2278@yahoo.com", "user_id": 42, "username": "raymond_henderson2278"},
"items": [{"item_id": 31, "product": {"name": "Designing Data-Intensive Applications
by Martin Kleppmann", "price": 54.99, "category": "Books", "product_id": 87},
"quantity": 1}, {"item_id": 32, "product": {"name": "High Performance MySQL by Baron
Schwartz", "price": 59.99, "category": "Books", "product_id": 90}, "quantity": 1},
{"item_id": 33, "product": {"name": "Database Internals by
Alex Petrov", "price": 54.99, "category": "Books", "product_id": 91}, "quantity":
1}], "status": "active", "_metadata": {"etag": "4ac9dee9b8415aeafb4b151deb6bc46a"}}'
where data->'$._id' = 12;
```

ERROR 6494 (HY000): Cannot update JSON duality view. The ETAG of the document in the database did not match the ETAG '"4ac9dee9b8415aeafb4b151deb6bc46a"' passed in.

DML

```
mysql> UPDATE shopping_cart_dml_dv SET data = '{"_id": 12, "user": {"email":
"r.henderson2278@yahoo.com", "user_id": 42, "username": "raymond_henderson2278"}, "items":
[{"item_id": 31, "product": {"name": "Designing Data-Intensive Applications by Martin Kleppmann",
"price": 54.80, "category": "Books", "product_id": 87}, "quantity": 1}, {"item_id": 32, "product":
{"name": "High Performance MySQL by Baron Schwartz", "price": 59.80, "category": "Books",
"product_id": 90}, "quantity": 1}, {"item_id": 33, "product": {"name": "Database Internals by
Alex Petrov", "price": 54.80, "category": "Books", "product_id": 91}, "quantity": 1}], "status":
"active"}' where data->'$._id' = 12;
Query OK, 3 rows affected, 1 warning (0.012 sec)
Rows affected: 3  Warnings: 1.

mysql> show warnings;
+---------+------+-------------------------------+
| Level   | Code | Message                       |
+---------+------+-------------------------------+
| Warning | 6493 | ETAG missing for updated value |
+---------+------+-------------------------------+
1 row in set (0.004 sec)
```

DML

```
mysql> insert into shopping_cart_dml_dv values ('{"_id": 100, "user": {"email":
"r.henderson2278@yahoo.com", "user_id": 42, "username": "raymond_henderson2278"}, "items":
[{"item_id":
 31, "product": {"name": "Designing Data-Intensive Applications by Martin Kleppmann", "price":
54.80, "category": "Books", "product_id": 87}, "quantity": 1}, {"item_id": 32, "product"
: {"name": "High Performance MySQL by Baron Schwartz", "price": 59.80, "category": "Books",
"product_id": 90}, "quantity": 1}, {"item_id": 33, "product": {"name": "Database Internals
by Alex Petrov", "price": 54.60, "category": "Books", "product_id": 91}, "quantity": 1}], "status":
"active"}');
```

Query OK, 7 rows affected (0.025 sec)
Rows affected: 7 Warnings: 0.

Explain

JDV are materialized and perform a full scan on the root table.

```
mysql> explain select * from shopping_cart_dml_dv where data->'$._id' = 10\G
*************************** 1. row ***************************
EXPLAIN: -> Filter: (json_extract(shopping_cart_dml_dv.`data`,'$._id') = 10)  (cost=3.85 rows=12)
    -> Table scan on shopping_cart_dml_dv  (cost=4.44..6.86 rows=12)
        -> Materialize  (cost=4.21..4.21 rows=12)
            -> Table scan on sc  (cost=1.45 rows=12)
            -> Select #5 (subquery in projection; dependent)
                -> Single-row index lookup on u using PRIMARY (user_id = sc.user_id)  (cost=0.35 rows=1)
            -> Select #3 (subquery in projection; dependent)
                -> Aggregate: json_arrayagg(json_duality_object( WITH (INSERT,UPDATE,DELETE)
'item_id':ci.item_id,'quantity':ci.quantity,'product':(select #4)))  (cost=1.6 rows=1)
                    -> Index lookup on ci using idx_cart_product (cart_id = sc.cart_id)  (cost=0.963
rows=2.75)
                -> Select #4 (subquery in projection; dependent)
                    -> Single-row index lookup on p using PRIMARY (product_id = ci.product_id)  (cost=0.35
rows=1)

1 row in set, 3 warnings (0.002 sec)
```

Use cases

- Small root tables kept in memory.
  - Active shopping carts.
  - Connected users.
- Retrieve the full set of documents.
  - Product categories.
- Remember:
  - No where filtering inside of the JDV.
  - No views.
  - No temporary tables.

Thank you!

I'm Pep Pla.

(And the cat is Mortimer Moriarty AKA Morty)

You can reach me at:

pep.pla@percona.com

@peppla