Mysql 8.0

Scalability and performance

Graph QL

- Authentication
- Windows function
- CTE
- Recursive CTE
- JSON aggregation operators
- JSON table

Authentication

- 3 plugins to authenticate a user:
  - mysql_native_password
  - sha256_password
  - caching_sha2_password

- *caching_sha2_password* by default (8.0.4)
- only for new Mysql user
  - new PIM installation impacted
  - need to wait for an update of “mysqlnd”

Windows function

A window function performs a calculation across a set of table rows that are somehow related to the current row.

But unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single output row.

PostgreSQL documentation

created_date	amount	tax
10/03/2017	1	0.1
10/03/2017	2	0.2
10/03/2017	3	0.3
24/04/2019	10	1
24/04/2019	10	1
24/04/2019	10	1

Total amount per date (without tax column)?

```sql
SELECT
    created_date,
    SUM(amount) as total_amount
FROM amount_table
GROUP BY created_date
ORDER BY created_date;
```

created_date	total_amount
10/03/2017	6
24/04/2019	30

Total amount per date (without tax column)?

```sql
SELECT
    created_date,
    SUM(amount) OVER (
     PARTITION BY created_date
    )as total_amount
FROM amount_table
ORDER BY created_date;
```

created_date	total_amount
10/03/2017	6
10/03/2017	6
10/03/2017	6
24/04/2019	30
24/04/2019	30
24/04/2019	30

Total amount per date (with tax column)?

```sql
SELECT
    created_date,
    SUM(amount) as total_amount,
    tax
FROM amount_table
GROUP BY created_date, tax
ORDER BY created_date;
```

created_date	total_amount	tax
10/03/2017	1	0.1
10/03/201	2	0.2
10/03/2017	3	0.3
24/04/2019		1
24/04/2019	10	1
24/04/20	10	1

Total amount per date (with tax column)?

```sql
SELECT
    created_date,
    SUM(amount) OVER (
     PARTITION BY created_date
    )as total_amount,
    tax,
FROM amount_table
ORDER BY created_date;
```

created_date	total_amount	tax
10/03/2017	6	0.1
10/03/2017	6	0.2
10/03/2017	6	0.3
24/04/2019	30	1
24/04/2019	30	1
24/04/2019	30	1

Cumulated amount per date?

```sql
SELECT
    created_date,
    SUM(amount) OVER (
     PARTITION BY created_date ORDER BY amount
    )as cumu_amount
FROM amount_table
ORDER BY created_date;
```

created_date	cumu_amount
10/03/2017	1
10/03/2017	3
10/03/2017	6
24/04/2019	10
24/04/2019	20
24/04/2019	30

row_number	test	duration	cumulated_duration	%_cumulated_duration	%_cumulated_number_tests
1	Test / behat-ce / features/datagrid/datagrid_views.feature:95 – Datagrid views	2 minutes, 42 seconds	2 minutes, 42 seconds	0.4%	0.1%
2	Test / behat-ce / features/product/filtering/filter_products_per_family.feature:25 – Filter products per family	2 minutes, 39 seconds	5 minutes, 21 seconds	0.7%	0.2%
3	Test / behat-ce / features/product/filtering/filter_products_per_number_fields.feature:11 – Filter products by number field	2 minutes, 33 seconds	7 minutes, 54 seconds	1.0%	0.2%
4	Test / behat-ce / features/product/filtering/filter_products_per_with_multiple_prices.feature:33 – Filter products with multiples prices filters	2 minutes, 18 seconds	10 minutes, 12 seconds	1.4%	0.3%
5	Test / behat-ce / features/import/product_model/import_and_compute_descendants.feature:39 – Create product models through CSV import and update their descendants	2 minutes, 13 seconds	12 minutes, 26 seconds	1.6%	0.4%

```sql
SELECT
  row_id as "row_number",
  test_name as "test",
  duration,
  cumulated_duration,
  cumulated_duration/total_duration as "%_cumulated_duration",
  row_id/total_number_tests as "%_cumulated_number_tests"
FROM (
  SELECT
    ROW_NUMBER() OVER(ORDER BY duration DESC) AS row_id,
    test_name,
    duration,
    SUM(duration) OVER (ORDER BY duration DESC) as cumulated_duration,
    SUM(duration) OVER () as total_duration,
    count(test_name) over() as total_number_tests
  FROM
    test_metric
  WHERE
    pipeline_name="$pipeline_name"
    AND branch_name = "$branch_name"
    AND run_id = "$run_id"
    AND type IN ($type)
  ORDER BY duration DESC
) metric;
```

In the PIM?

- No use case for now
- But useful for reporting: KPI?

Common Table Expression (CTE)

Derived table

```sql
SELECT c.id, c.code
FROM (
  SELECT *
  FROM pim_catalog_category c
  WHERE c.parent_id IS NULL
) as root_category;
```

CTE

```sql
WITH root_category AS
  (
      SELECT *
      FROM pim_catalog_category c
      WHERE c.parent_id IS NULL
  )
SELECT c.id, c.code FROM root_category c;
```

Recursive CTE

I want all category paths

```sql
WITH RECURSIVE category_tree AS
    (
        SELECT c.id, c.code, c.code as path
        FROM pim_catalog_category c
        WHERE c.parent_id IS NULL
        UNION ALL
        SELECT c.id, c.code, CONCAT(t.path, '/', c.code) as path
        FROM pim_catalog_category c JOIN category_tree t on c.parent_id = t.id
    )
SELECT * FROM category_tree;
```

id	code	path
1	master	master
2	cameras	master/cameras
3	audio	master/audio_video

I want all category paths

STEP 1

```sql
WITH RECURSIVE category_tree AS
    (
        SELECT c.id, c.code, c.code as path
        FROM pim_catalog_category c
        WHERE c.parent_id IS NULL
        UNION ALL
        SELECT c.id, c.code, CONCAT(t.path, '/', c.code) as path
        FROM pim_catalog_category c JOIN category_tree t on c.parent_id = t.id
    )
SELECT * FROM category_tree;
```

id	code	path
1	master	master

I want all category paths

STEP 2

```sql
WITH RECURSIVE category_tree AS
    (
        SELECT c.id, c.code, c.code as path
        FROM pim_catalog_category c
        WHERE c.parent_id IS NULL
        UNION ALL
        SELECT c.id, c.code, CONCAT(t.path, '/', c.code) as path
        FROM pim_catalog_category c JOIN category_tree t on c.parent_id = t.id
    )
SELECT * FROM category_tree;
```

id	code	path
2	cameras	master/cameras
3	audio	master/audio_video

I want all category paths

STEP 3

```sql
WITH RECURSIVE category_tree AS
    (
        SELECT c.id, c.code, c.code as path
        FROM pim_catalog_category c
        WHERE c.parent_id IS NULL
        UNION ALL
        SELECT c.id, c.code, CONCAT(t.path, '/', c.code) as path
        FROM pim_catalog_category c JOIN category_tree t on c.parent_id = t.id
    )
SELECT * FROM category_tree;
```

id	code	path

In the PIM?

- Category tree
- Product models and variant products

```sql
WITH RECURSIVE complete_variant_products AS
(
    SELECT p.identifier, p.product_model_id as parent_id, p.raw_values
    FROM pim_catalog_product p
    WHERE p.product_model_id IS NOT NULL
    UNION ALL
    SELECT vp.identifier, pm.parent_id, JSON_MERGE_PATCH(pm.raw_values, vp.raw_values)
    FROM pim_catalog_product_model pm JOIN complete_variant_products vp on pm.id =
vp.parent_id
)
SELECT identifier, raw_values FROM complete_variant_products WHERE parent_id IS NULL
```

identifier	raw_values
1	{"sku": {"<all_channels>": {"<all_locales>": "1111111111"}}, "name": {"<all_channels>": {"<all_locales>": "Heritage jacket navy"}}, "size": {"<all_channels>": {"<all_locales>": "xxl"}}, "color": {"<all_channels>": {"<all_locales>": "blue"}}, "price": {"<all_channels>": {"<all_locales>": [{"amount": "999.00", "currency": "EUR"}, {"amount": null, "currency": "USD"}]}}, "erp_name": {"<all_channels>": {"en_US": "Amor"}}, "supplier": {"<all_channels>": {"<all_locales>": "zaro"}}, "collection": {"<all_channels>": {"<all_locales>": ["summer_2016"]}}, "description": {"ecommerce": {"en_US": "Heritage jacket navy blue tweed suit with single breasted 2 button. 53% wool, 22% polyester, 18% acrylic, 5% nylon, 1% cotton, 1% viscose. Dry Cleaning uniquement.Le mannequin measuring 1m85 and wears UK size 40, size 50 FR"}}, "wash_temperature":

JSON aggregation operators

I want categories with labels

```sql
SELECT
  c.code, ct.locale, ct.label
FROM pim_catalog_category c
LEFT JOIN pim_catalog_category_translation ct on ct.foreign_key = c.id;
```

category	locale	label
master	en_US	master US
master	fr_FR	master FR
acer	en_US	acer US
acer	fr_FR	acer FR

I want categories with labels

```sql
SELECT
       c.code,
       JSON_OBJECT(
               'locale', ct.locale,
               'label', ct.label
   ) as translations
   FROM pim_catalog_category c
   LEFT JOIN pim_catalog_category_translation ct on ct.foreign_key = c.id;
```

category	translations
master	{“locale”:”en_US”, “label”: “master US”}
master	{“locale”:”fr_FR”, “label”: “master FR”}
acer	{“locale”:”en_US”, “label”: “acer US”}
acer	{“locale”:”fr_FR”, “label”: “acer US”}

I want categories with labels

```sql
SELECT
    c.code,
    JSON_ARRAYAGG(
        JSON_OBJECT(
            'locale', ct.locale,
            'label', ct.label
        )
    ) as translations
FROM pim_catalog_category c
LEFT JOIN pim_catalog_category_translation ct on ct.foreign_key = c.id
GROUP BY c.code;
```

category	translations
master	[{“label”: “master US”, “locale”: “en_US”}, {“label”: “master FR”, “locale”: “fr_FR”}]
acer	[{“label”: “master US”, “locale”: “en_US”}, {“label”: “master FR”, “locale”: “fr_FR”}]

JSON table

I want all attributes of a product

```sql
SELECT
   p.identifier,
   js.attribute
FROM
   pim_catalog_product p,
   JSON_TABLE (
      JSON_KEYS(p.raw_values),
        "$[*]" COLUMNS (
           attribute VARCHAR(50) PATH "$"
        )
   ) as js;
```

identifier	attribute
tvsam32	sku
tvsam32	size
Biker-jacket	sku
Biker-jacket	picture

But

- Does not work with keys
- No useable with our current representation of “raw_values”

identifier	attribute	channel	locale	data
tvsam32	sku	null	null	tvsam32
tvsam32	size	ecommerce	null	7
Biker-jacket	sku	null	null	Biker-jacket
Biker-jacket	picture	ecommerce	en_US	/my_picture

And in other RDMS ?

RDMS	Windows Function	CTE	JSON Aggregation	JSON Table
PostgreSQL	✓	✓	✓	✓
MariaDB	✓	✓	✗	✗
Oracle	✓	✓	✓	✓

End

https://github.com/ahocquard/akeneo-mysql