Mysql 8.0

Scalability and performance

Graph QL

Performance

What are the objectives?

- Generally a maximum allowed response time
  - for the export
  - for the datagrid
  - for the PEF
  - for the API
  - etc
- A maximal server cost?

How?

- Given defined **axes**
  - given a dataset (X products, X families, etc)
  - given a machine (RAM, number of servers)
  - given a number of concurrent users
  - etc…

Performance

Only **you** can determine what is acceptable

Example 1:

- Product grid displayed in 1 second, server at 30$/month
- Product grid displayed in 2 second, server at 30$/month

Example 2:

- Product grid displayed in 1 second, server at 30$/month
- Product grid displayed in 1 second, server at 100$/month

Scalability

How what you measure is impacted when you change the axes?

What are the limits of the axes to fulfil the objectives?

In theory, you should change only one axis at a time.

- increase the number of products
- increase the size of the products
- increase the number categories
- increase the number of concurrent user
- decrease the number of front servers
- decrease the RAM of the server
- etc

Scalability

Be careful to define correctly what are the objectives and what are the axis.

- If the number of products is an objective

  The PIM should handle 10 million products.
  Datagrid should be displayed in less than 1 second.

- If the number of products is an axis

  How many products can the PIM handle to display the datagrid in less than 1 second?

Scalability

We should determine the objectives and then the axes.

Scalability is specialisation: the more you have objectives, harder it is.

Goal of the Proof of concept:

- Improve the response time of the PIM
- Reduce the pressure on the database
- Read only (API, export, UI)

Why the PIM is slow when exporting data?

Too many database requests

With the API (community edition):

- 100 products
- 352 Product values/product
- 2264 SQL requests!!!
- ~6.5 seconds

- 100 families
- 100 attributes/family
- 321 SQL requests
- ~2.5 seconds

Lazy loading

Why so many requests?

- ORM + lazy loading = n + 1 requests

Explanations for 100 families with the API (321 requests):

```php
public function normalize($family, $format = null, array $context = [])
{
    return [
        'code'                   => $family->getCode(),
        'attributes'             => $this->normalizeAttributes($family),
        'attribute_as_label'     => null !== $family->getAttributeAsLabel()
            ? $family->getAttributeAsLabel()->getCode() : null,
        'attribute_as_image'     => null !== $family->getAttributeAsImage()
            ? $family->getAttributeAsImage()->getCode() : null,
        'attribute_requirements' => $this->normalizeRequirements($family),
        'labels'                 => $this->translationNormalizer-
>normalize($family, 'standard', $context),
    ];
}
```

- 1 request for to get all the families
- 1 request/family to get attributes = 100 requests
- 1 request/family to get translations = 100 requests
- 1 request/family to get requirements = 100 requests
- other stuff (authentication, event listener)

Lazy loading

Pros:

- Easy to implement
- Avoid useless SQL call
- Avoid data redundancy (cf eager loading)

Cons:

- more entities, more requests
- more properties linked to other entities, more requests
- useless hydration by Doctrine/ORM: not all properties are useful in the API format

Lazy loading is an SQL JOIN done by your application code

Eager loading

```sql
SELECT
     f.code,
    a_label.code as label_code,
     a_image.code as image_code
FROM pim_catalog_family f
 LEFT JOIN pim_catalog_attribute a_label on a_label.id = f.label_attribute_id
 LEFT JOIN pim_catalog_attribute a_image on a_image.id = f.image_attribute_id
 where f.code IN ('family_1', 'family_2);
```

family_code	label_code	image_code
family_1	a_label_1	a_image_1
family_2	a_label_2	a_image_2

Eager loading

```sql
SELECT
     f.code,
     a_label.code as label_code,
     a_image.code as image_code,
     a.code as attribute_code
FROM pim_catalog_family f
 LEFT JOIN pim_catalog_attribute a_label on a_label.id = f.label_attribute_id
 LEFT JOIN pim_catalog_attribute a_image on a_image.id = f.image_attribute_id
 LEFT JOIN pim_catalog_family_attribute fa on fa.family_id = f.id
 LEFT JOIN pim_catalog_attribute a on a.id = fa.attribute_id
 WHERE f.code IN ('family_1', 'family_2);
```

family_code	label_code	image_code	attribute_code
family_1	a_label_1	a_image_1	a_1
family_1	a_label_1	a_image_1	a_2
family_2	a_label_2	a_image_2	a_3
family_2	a_label_2	a_image_2	a_4

Eager loading

```sql
SELECT
     f.code,
     a_label.code as label_code,
     a_image.code as image_code,
     a.code as attribute_code,
      c.code as channel_code,
     a_requi.code as attribute_requirement_code
FROM pim_catalog_family f
 LEFT JOIN pim_catalog_attribute a_label on a_label.id = f.label_attribute_id
 LEFT JOIN pim_catalog_attribute a_image on a_image.id = f.image_attribute_id
 LEFT JOIN pim_catalog_family_attribute fa on fa.family_id = f.id
 LEFT JOIN pim_catalog_attribute a on a.id = fa.attribute_id
 LEFT JOIN pim_catalog_attribute_requirement r on r.family_id = f.id
 LEFT JOIN pim_catalog_channel c on c.id = r.channel_id and r.required = '1'
 LEFT JOIN pim_catalog_attribute a_requi on a_requi.id = r.attribute_id

 WHERE f.code IN ('family_1', 'family_2);
```

family_code	label_code	image_code	attribute_code	channel_code	attribute_requireme
family_1	a_label_1	a_image_1	a_1	ecommerce	a_1
family_1	a_label_1	a_image_1	a_1	tablet	a_1
family_1	a_label_1	a_image_1	a_1	ecommerce	a_2
family_1	a_label_1	a_image_1	a_1	tablet	a_2
family_1	a_label_1	a_image_1	a_2	ecommerce	a_1
family_1	a_label_1	a_image_1	a_2	tablet	a_1
family_1	a_label_1	a_image_1	a_2	ecommerce	a_2
family_1	a_label_1	a_image_1	a_2	tablet	a_2
family_2	a_label_2	a_image_2	a_3	ecommerce	a_3
family_2	a_label_2	a_image_2	a_3	tablet	a_3
family_2	a_label_2	a_image_2	a_3	ecommerce	a_4
family_2	a_label_2	a_image_2	a_3	tablet	a_4
family_2	a_label_2	a_image_2	a_4	ecommerce	a_3
family_2	a_label_2	a_image_2	a_4	tablet	a_3
family_2	a_label_2	a_image_2	a_4	ecommerce	a_4
family_2	a_label_2	a_image_2	a_4	tablet	a_4

Eager loading

Pros:

- Easy to implement
- One request
- Not costly for 1-1 relations

Cons:

- redundant data with 1-n relations
- combinatorial explosion with JOIN
- very costly to flatten the data
- it does not scale at all

Eager loading

Icecat catalog:

- 17 families
- 4599 lines returned
- ~15 ms

80% catalog:

- 125 families
- 100 attributes/family
- ~ 2 millions of lines returned!!!
- ~ 4 seconds

Eager loading

Do you think that Doctrine/ORM in PHP7 can flatten 2 millions of lines quickly?

Reminder

I want categories with labels

```
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

```
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”}]

Eager loading

Let’s do it with Mysql:

```sql
SELECT
    f.code,
    a_label.code as attribute_as_label_code,
    a_image.code as attribute_as_image_code,
    c.code as channel_code,
    JSON_ARRAYAGG(a.code) as attribute_code,
    JSON_ARRAYAGG(a_requi.code) as requirements
FROM pim_catalog_family f
LEFT JOIN pim_catalog_family_attribute fa on fa.family_id = f.id
LEFT JOIN pim_catalog_attribute a on a.id = fa.attribute_id
LEFT JOIN pim_catalog_attribute a_label on a_label.id = f.label_attribute_id
LEFT JOIN pim_catalog_attribute a_image on a_image.id = f.image_attribute_id
LEFT JOIN pim_catalog_attribute_requirement r on r.family_id = f.id
LEFT JOIN pim_catalog_channel c on c.id = r.channel_id and r.required = '1'
LEFT JOIN pim_catalog_attribute a_requi on a_requi.id = r.attribute_id
GROUP BY f.code, c.code;
```

family_code	label_code	image_code	attribute_code	channel	attribute_requirements
family_1	a_label_1	a_image_1	[“a_1”, “a_1”, “a_2”, “a_2”]	ecommerce	[“a_1”, “a_1”, “a_2’,“a_2”]
family_1	a_label_1	a_image_1	[“a_2”, “a_2”, “a_3”, “a_3”]	tablet	[“a_1”, “a_1”, “a_2’,“a_2”]
family_2	a_label_2	a_image_2	[“a_2”, “a_2”, “a_3”, “a_3”]	ecommerce	[“a_2”, “a_2”, “a_3”, “a_3”]
family_2	a_label_2	a_image_2	[“a_2”, “a_2”, “a_3”, “a_3”]	tablet	[“a_2”, “a_2”, “a_3”, “a_3”]

Icecat catalog:

~ 100 ms

80% catalog:

Timeout

> 30 sec

Eager loading

Given that Mysql can’t do that efficiently, do you think that Doctrine/ORM in PHP7 can flatten 2 millions of lines quickly?

Solutions

- Lazy loading
  - intensive for the DB
  - it works slowly with high volume
- Eager loading
  - it does not work with high volume
- Cache
  - invalidation of cache
  - hard in the PIM (graph dependency)
- Projection
  - invalidation of projection
  - hard in the PIM (graph dependency)
  - probably ok for the product
- and…

Mysql aggregation

```sql
SELECT
    f.code,
    a_label.code as attribute_as_label_code,
    a_image.code as attribute_as_image_code,
    c.code as channel_code,
    JSON_ARRAYAGG(a.code) as attribute_code,
    JSON_ARRAYAGG(a_requi.code) as requirements
FROM pim_catalog_family f
LEFT JOIN pim_catalog_family_attribute fa on fa.family_id = f.id
LEFT JOIN pim_catalog_attribute a on a.id = fa.attribute_id
LEFT JOIN pim_catalog_attribute a_label on a_label.id = f.label_attribute_id
LEFT JOIN pim_catalog_attribute a_image on a_image.id = f.image_attribute_id
LEFT JOIN pim_catalog_attribute_requirement r on r.family_id = f.id
LEFT JOIN pim_catalog_channel c on c.id = r.channel_id and r.required = '1'
LEFT JOIN pim_catalog_attribute a_requi on a_requi.id = r.attribute_id
GROUP BY f.code, c.code;
```

- create 2 millions of line
- then, flatten the result
- inefficient

Mysql aggregation

```sql
SELECT
    f.code,
    a_label.code as attribute_as_label_code,
    a_image.code as attribute_as_image_code,
    c.code as channel_code,
    JSON_ARRAYAGG(a.code) as attribute_code,
    JSON_ARRAYAGG(a_requi.code) as requirements
FROM pim_catalog_family f
LEFT JOIN pim_catalog_family_attribute fa on fa.family_id = f.id
LEFT JOIN pim_catalog_attribute a on a.id = fa.attribute_id
LEFT JOIN pim_catalog_attribute a_label on a_label.id = f.label_attribute_id
LEFT JOIN pim_catalog_attribute a_image on a_image.id = f.image_attribute_id
LEFT JOIN pim_catalog_attribute_requirement r on r.family_id = f.id
LEFT JOIN pim_catalog_channel c on c.id = r.channel_id and r.required = '1'
LEFT JOIN pim_catalog_attribute a_requi on a_requi.id = r.attribute_id
GROUP BY f.code, c.code;
```

The trick is to aggregate data after each JOIN.

Mysql aggregation

```sql
SELECT
     f.code,
     a_label.code as label_code,
     a_image.code as image_code
FROM pim_catalog_family f
 LEFT JOIN pim_catalog_attribute a_label on a_label.id = f.label_attribute_id
 LEFT JOIN pim_catalog_attribute a_image on a_image.id = f.image_attribute_id
 where f.code IN ('family_1', 'family_2);
```

family_code	label_code	image_code
family_1	a_label_1	a_image_1
family_2	a_label_2	a_image_2

Mysql aggregation

```sql
SELECT
    family.*,
    JSON_ARRAYAGG(a.code) as attribute_codes
FROM (
    SELECT
        f.id,
        f.code,
        a_label.code as label_code,
        a_image.code as image_code
    FROM pim_catalog_family f
    LEFT JOIN pim_catalog_attribute a_label on a_label.id = f.label_attribute_id
    LEFT JOIN pim_catalog_attribute a_image on a_image.id = f.image_attribute_id
   WHERE f.code IN ('family_1', 'family_2)
) as family
LEFT JOIN pim_catalog_family_attribute fa on fa.family_id = family.id
LEFT JOIN pim_catalog_attribute a on a.id = fa.attribute_id
GROUP BY family.code;
```

family_code	label_code	image_code	attribute_code
family_1	a_label_1	a_image_1	[“a_1”, “a_2”]
family_2	a_label_2	a_image_2	[“a_2”, “a_3”]

Mysql aggregation

```sql
SELECT
     family_with_attributes.*,
    c.code as channel,
    JSON_ARRAYAGG(a_requi.code) as a_requi
FROM (
     SELECT
          family.*,
          JSON_ARRAYAGG(a.code) as attribute_codes
     FROM (
          SELECT
               f.id,
               f.code,
               a_label.code as label_code,
               a_image.code as image_code
          FROM pim_catalog_family f
          LEFT JOIN pim_catalog_attribute a_label on a_label.id = f.label_attribute_id
          LEFT JOIN pim_catalog_attribute a_image on a_image.id = f.image_attribute_id
     ) as family
     LEFT JOIN pim_catalog_family_attribute fa on fa.family_id = family.id
     LEFT JOIN pim_catalog_attribute a on a.id = fa.attribute_id
     GROUP BY family.code
) as family_with_attributes
LEFT JOIN pim_catalog_attribute_requirement r on r.family_id = family_with_attributes.id
LEFT JOIN pim_catalog_channel c on c.id = r.channel_id and r.required = '1'
LEFT JOIN pim_catalog_attribute a_requi on a_requi.id = r.attribute_id
GROUP BY family_with_attributes.id, c.code;
```

family_code	label_code	image_code	attribute_code	channel	a_requi
family_1	a_label_1	a_image_1	[“a_1”, “a_2”]	tablet	[“a_1”, “a_2”]
family_1	a_label_1	a_image_1	[“a_1”, “a_2”]	ecommerce	[“a_1”, “a_2”]
family_2	a_label_2	a_image_2	[“a_2”, “a_3”]	tablet	[“a_2”, “a_3”]
family_2	a_label_2	a_image_2	[“a_2”, “a_3”]	ecommerce	[“a_2”, “a_3”]

Mysql aggregation

```sql
SELECT
      complete_family.id,
      complete_family.code,
      complete_family.label_code,
      complete_family.image_code,
      complete_family.attribute_codes,
      JSON_OBJECTAGG(complete_family.channel, complete_family.a_requi) as requirements
FROM (
      SELECT
            family_with_attributes.*,
            c.code as channel,
            JSON_ARRAYAGG(a_requi.code) as a_requi
      FROM (
            SELECT
                  family.*,
                  JSON_ARRAYAGG(a.code) as attribute_codes
            FROM (
                  SELECT
                        f.id,
                        f.code,
                        a_label.code as label_code,
                        a_image.code as image_code
                  FROM pim_catalog_family f
                  LEFT JOIN pim_catalog_attribute a_label on a_label.id = f.label_attribute_id
                  LEFT JOIN pim_catalog_attribute a_image on a_image.id = f.image_attribute_id
            ) as family
            LEFT JOIN pim_catalog_family_attribute fa on fa.family_id = family.id
            LEFT JOIN pim_catalog_attribute a on a.id = fa.attribute_id
            GROUP BY family.code
      ) as family_with_attributes
      LEFT JOIN pim_catalog_attribute_requirement r on r.family_id = family_with_attributes.id
      LEFT JOIN pim_catalog_channel c on c.id = r.channel_id and r.required = '1'
      LEFT JOIN pim_catalog_attribute a_requi on a_requi.id = r.attribute_id
      GROUP BY family_with_attributes.id, c.code
      ) as complete_family
GROUP BY complete_family.id;
```

family_code	label_code	image_code	attribute_code	requirements
family_1	a_label_1	a_image_1	[“a_1”, “a_2”]	[“tablet”: [“a_1”, “a_2”], “ecommerce” : [“a_1”,
family_2	a_label_2	a_image_2	[“a_2”, “a_3”]	[“tablet”: [“a_1”, “a_2”], “ecommerce” : [“a_1”,

Mysql aggregation

- hydration friendly
- only returns what you want
- less pressure on database
- less network traffic with the database
- almost the API format!

Mysql aggregation

100 families with API:

Before

- 100 families
- 100 attributes/family
- ~ 2,5 seconds
- 321 requests (with 21 from other stuff)

After

- 100 families
- 100 attributes/family
- ~ 400 ms
- 22 requests (with 21 from other stuff)

Mysql aggregation

- increase number of families: still 1 request
- increase number of attributes/family: still 1 request
- constant number of request = good scaling

Mysql aggregation

100 families with API:

Before

- 100 families
- 100 attributes/family
- ~ 2,5 seconds
- 321 requests (with 21 from other stuff)

After

- 100 families
- 100 attributes/family
- ~ 400 ms
- 22 requests (with 21 from other stuff)

Mysql aggregation

100 products with API:

Before

- 100 products
- 360 PV/products
- ~ 7 seconds

After

- 100 products
- 360 PV/products
- ~ 700 ms, ~400 ms coming from PHP normalisation

Hydration

```php
class Family
{
    /** @var FamilyCode */
    private $code;

    /** @var \DateTimeInterface */
    private $created;

    /** @var \DateTimeInterface */
    private $updated;

    /** @var AttributeCode */
    private $attributeAsLabelCode;

    /** @var AttributeCode */
    private $attributeAsImageCode;

    /** @var AttributeCode[] */
    private $attributeCodes = [];

    /** @var AttributeRequirement[] */
    private $attributeRequirements = [];

    /** @var FamilyLabel[] */
    private $labels = [];
```

Hydration

```php
foreach ($rows as $row) {
    $code = Type::getType(Type::STRING)->convertToPhpValue($row['code'], $platform);
    $created = Type::getType(Type::DATETIME)->convertToPhpValue($row['created'], $platform);
    $updated = Type::getType(Type::DATETIME)->convertToPhpValue($row['updated'], $platform);
    $attributeAsLabelCode = Type::getType(Type::STRING)->convertToPhpValue($row['attribute_as_label_code'],
$platform);
    $attributeAsImageCode = Type::getType(Type::STRING)->convertToPhpValue($row['attribute_as_image_code'],
$platform);

    $families[] = new Family(
        FamilyCode::createFromString($code),
        $created,
        $updated,
        null !== $attributeAsLabelCode ? AttributeCode::createFromString($attributeAsLabelCode) : null,
        null !== $attributeAsImageCode ? AttributeCode::createFromString($attributeAsImageCode) : null,
        $this->hydrateAttributeCodes($row),
        $this->hydrateAttributeRequirements($row),
        $this->hydrateLabels($row)
    );
}
```

Hydration

```php
private function hydrateAttributeCodes(array $row): array
{
    $attributeCodes = [];
    if (isset($row['attribute_codes'])) {
        $decodedAttributeCodes = json_decode($row['attribute_codes'], true);
        if (null !== $decodedAttributeCodes) {
            foreach ($decodedAttributeCodes as $attributeCode) {
                $attributeCodes[] =
AttributeCode::createFromString($attributeCode);
            }
        }

        return $attributeCodes;
}
```

Hydration

- hydration is not very hard to implement
- easy to spec
- easy to do integration tests
- easy to implement performance tests on the query!

In the PIM

- improve drastically performance without too many changes
  - API
  - export
  - UI
- prevent workarounds
  - getting 5000 products in the association ;)
  - better for batching (say goodbye to memory leaks in export)
  - better scaling for the number of users
    - less requests
    - less network traffic
  - reduce the number of database needed for the onboarder? #money
  - last but not least: 70% stuff is done in CE
    - tested
    - “speced"

In the PIM

For the products, it would be better to use ES projection.

I agree. But how do you generate projections?

End

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

https://github.com/ahocquard/pim-community-dev/tree/poc-products/src/Pim/Bundle/ResearchBundle

(sorry for the bundle name)