MAGENTO 2.0

INDEXATION TECHNIQUES FOR HIGH PERFORMANCE

by Ivan Chepurnyi

WHAT IS A MAGENTO INDEX?

MAGENTO INDEX

- Represented as a separate table
- Optimized for retrieval, filtration and/or sorting.
- Just a snapshot of data at some point in time.

WHEN TO CREATE IT?

DATA FILTRATION

- Product Visibility
- Product Status
- Category Relation

DATA SORTING

- Category Position
- Bestsellers
- Stock Quanttities

COMPLEX LOGIC CACHE

- Stock Availability
- Product Price
- Anchor Category Relation

WHEN DO NOT TO USE THEM?

FOR DATA RETRIEVAL

It does not make sense to flatten EAV structure for retrieval

WHEN ACCURATE DATA IS CRITICAL

Using price or stock index in shopping cart is a bad idea

HIGH AVAILABILITY INDEXATION CHALLANGES

TABLE LOCKS

Index generation can create a lock on table you don't expect

```
UDPATE some_index_table index
    INNER JOIN catalog_product_entity_varchar product_data ON ...
        SET index.some_value = product_data.value
```

SOLUTION

- Avoid cross table updates on live tables
- Create snapshot of data for your index

MYSQL MEMORY

MySQL memory consuption is increasing with amount of records you need to update.

These are known MySQL serial killers:

```
INSERT INTO .. SELECT ... FROM
```

```
UPDATE ... JOIN ... SET ...
```

SOLUTION

Range your updates into smaller data-sets

INDEX DOWNTIME

During re-indexation there is a step when index table is locked and no data is available in it.

SOLUTION

- Create two identical index tables
- Re-index only one at a time
- Switch one to use on the frontend via configuration model

DID ANYONE CREATED AN INDEXER IN MAGENTO 1.X?

It was a nightmare comparing to Magento 2.0...

IT IS NOW INCREDIBLY SIMPLE...

ONLY TWO CONFIGURATION FILES REQUIRED, AND NO EVENT MATCHING MAGIC:

`indexer.xml` and `mview.xml`

MVIEW.XML

Allows you to subscribe to changes in any table

`Magento\Framework\Mview\ActionInterface`

```
<view id="mview_id" class="MviewClass" group="indexer">
    <subscriptions>
        <table name="table_to_check"
                    entity_column="primary_key" />
    </subscriptions>
</view>
```

WHY IS IT CALLED MVIEW?

INDEXER.XML

Just a regular Magento indexer definition

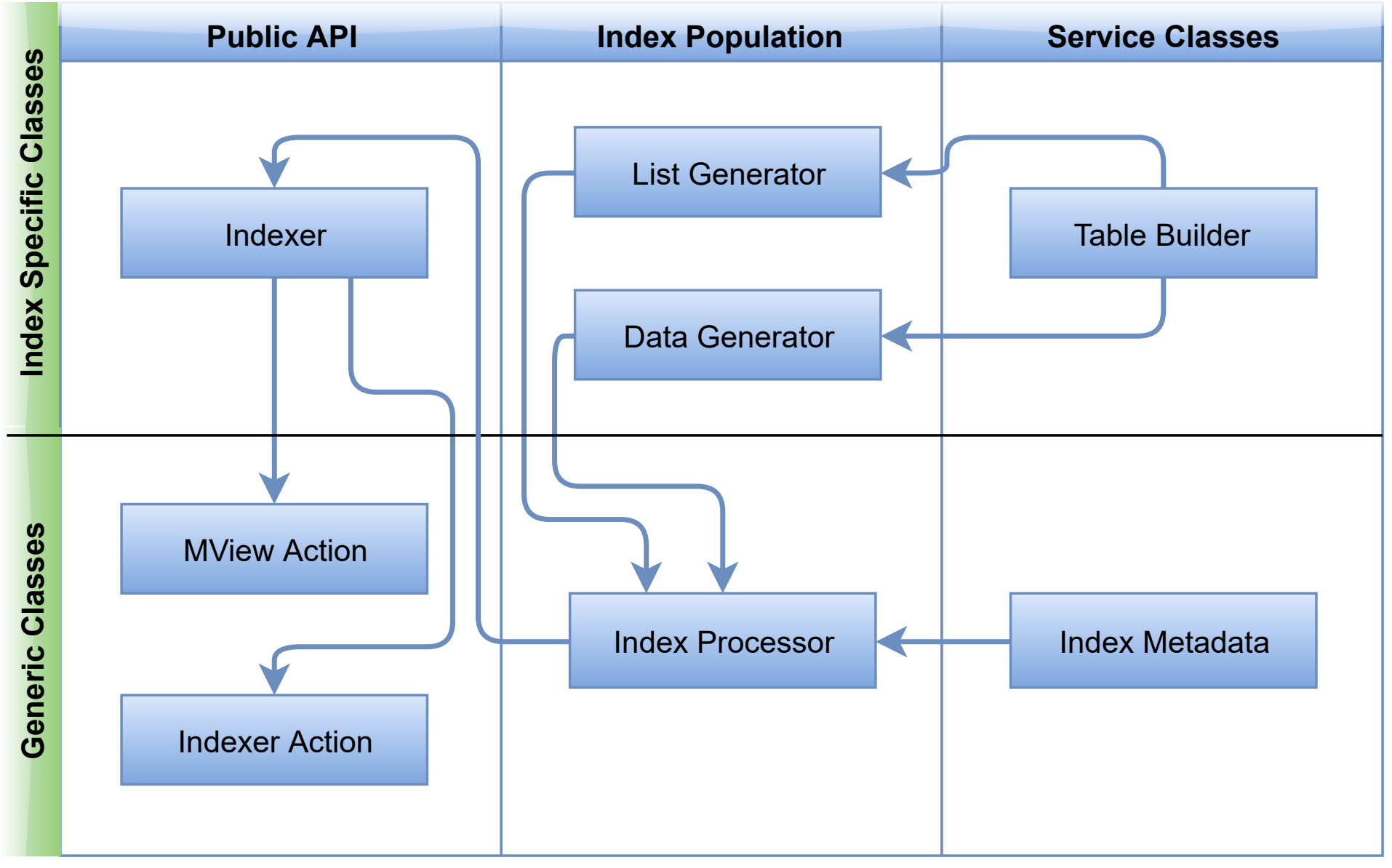
`Magento\Framework\Indexer\ActionInterface`

```
<indexer id="indexer_code" view_id="mview_code"
         class="YourIndexerClass">
    <title translate="true">Your Title</title>
    <description translate="true">Your Description</description>
</indexer>
```

So how to implement HA index in Magento 2.0?

I really like re-usable components

So I come up with such indexer structure


Public API
Index Population
Service Classes
Index Specific Classes
Generic Classes
Indexer
List Generator
Table Builder
Data Generator
MView Action
Index Processor
Index Metadata
Indexer Action

INDEXER

- Generic interface for usage in MView and Indexer Action
- Has knowledge about the index structure
- Creates index conditions for processor

SAMPLE INTERFACE

```
interface IndexerInterface
{
    public function executeFull();

    public function executeByIds(array $ids);
}
```

LIST GENERATOR

- Creates fundation for our indexer
- Consists mostly from entity identifiers

SAMPLE INTERFACE

```
use Magento\Framework\Db\Ddl\Table;

interface ListGeneratorInterface
{
    /**
     * @return Table
     */
    public function generate(ConditionInterface $condition);
}
```

INDEX CONDITION

- Decouples creation and applying of condition

SAMPLE INTERFACE

```
use Magento\Framework\DB\Select;

interface ConditionInterface
{
    public function apply(
        Select $select,
        $alias,
        array $columnMap = []
    );
}
```

DATA GENERATOR

- Creates final subset of index data
- Data is based on List Generator data

SAMPLE INTERFACE

```
interface DataGeneratorInterface
{
    /**
     * @return Table
     */
    public function generate(Table $listTable);
}
```

INDEX PROCESSOR

- Recieves list of index conditions from indexer
- Decides how to populate an index table
- Toggles index table via metadata instance

SAMPLE INTERFACE

```
interface ProcessorInterface
{
    public function process($conditions, $live = false);
}
```

INDEX METADATA

- Controls which table to use for indexer and which for frontend
- Should be used in your entity repository to filter data by it
- Provides usefull attribute information

SAMPLE INTERFACE

```
interface MetadataInterface
{
    public function getTableToIndex($live = false);

    public function getIndexTable();

    public function toggleTable();

    public function getAttributesInfo($entityType, $attributeCodes);
}
```

REAL INDEX EXAMPLE

Created in 3 hours during the workshop on the first #DevParadise day

http://bit.ly/m2-boilerplate

WANT TO LEARN MORE ABOUT THESE TECHNIQUES?

Catch me at coffee break or during after party!

Q&A

@IvanChepurnyi

ivan@ecomdev.org