MSEARCH-320 Technical approach to browse Contributors

#1 Browsing by Contributors using dedicated Elasticsearch index

This approach is the same as browsing by contributors, but with a few extensions.

Pros	Cons
Requires only Elasticsearch and Kafka	Requires additional space to store the dedicated index with all linked instance ids
	Requires integration with kafka to prevent race conditions and get rid of optimistic locks on Elasticsearch documents
	Requires additional code to manage update and delete events

Approach is contains a several steps to implement:

1. Send contributor events from instance to the search.instance-contributor topic (message key = sha1 hash of concatenatination - tenantId + contributorTypeNameId + name)
2. Implement a new Kafka Listener to listen contributor only events (Kafka will arrange contributors with the same key in the same partition, allowing event processing without optimistic locking)
3. Implement a new InstanceContributor repository with following upsert request for events groups
4. Implement a query which allows to find contributor exact by name and type (it can be CQL with double equal sign, but it looks like that **John** and **J ohn.** are treated as the same value (ES standard tokenizer))

Upsert query example:

```
{
  "script": {
    "source": "def set=new LinkedHashSet(ctx._source.instances);set.addAll(params.ins);params.del.forEach(set::
remove);ctx._source.instances=set",
    "lang": "painless",
    "params": {
      "ins": [ "instanceId#1|contributorTypeId#1", "instanceId#2|contributorTypeId#1",
"instanceId#3|tcontributorTypeId#1" ],
      "del": [ "instanceId#4|contributorTypeId#1"]
    }
  },
  "upsert": {
    "id": "abshc",
    "name": "Antoniou, Grigoris",
    "contributorTypeNameId": "contriboturTypeNameId",
    "instances": [ "instanceId#1|contributorTypeId#1", "instanceId#2|contributorTypeId#1",
"instanceId#3|contributorTypeId#1" ]
  }
}
```

#2 Browsing by Contributors using PostgreSQL table

This options provides ability to browse by using PostgreSQL table with index on the contributors field.

Database schema

```
create table instance_contributor
(
    contributor text not null,
    instance_id text not null,
    constraint instance_contributors_pk
        primary key (contributor, instance_id)
);

create index instance_contributors_contributor
    on diku_mod_search.instance_contributors (lower(contributor));
```

Insertions can be done in batch, which can be done configuring Spring Data Jpa:

Insert script

```
insert into instance_contributor(instance_id, contributor) values (?,?) on conflict do nothing;
```

Required configuration

```
spring:
  jpa:
    properties:
      hibernate:
        order_inserts: true
        order_updates: true
        jdbc.batch_size: 500
```

Java Entities

```
@Data
@Entity
@NoArgsConstructor
@Table(name = "instance_contributors")
@AllArgsConstructor(staticName = "of")
@SQLInsert(sql = "insert into instance_contributors(instance_id, contributor) values (?, ?) on conflict do
nothing")
public class InstanceContributorEntity implements Persistable<InstanceContributorEntityId> {

  @EmbeddedId
  private InstanceContributorEntityId id;

  @Override
  public boolean isNew() {
    return true;
  }
}

@Data
@Embeddable
@NoArgsConstructor
@AllArgsConstructor(staticName = "of")
public class InstancecontributorEntityId implements Serializable {

  private String contributor;
  private String instanceId;
}
```

Preceding Query

```
select contributor, count(*)
from instance_contributors
where contributor in (
  select distinct on (lower(contributor)) contributor
  from instance_contributors
  where lower(contributor) < :anchor
  order by lower(contributor) desc
  limit :limit
)
group by contributor
order by lower(contributor);
```

Succeeding Query

```
select contributor, count(*)
from instance_contributors
where contributor in (
  select distinct on (lower(contributor)) contributor
  from instance_contributors
  where lower(contributor) >= :anchor
  order by lower(contributor)
  limit :limit
)
group by contributor
order by lower(contributor);
```

Pros	Cons
Fast to query (faster than other options)	Requires additional space to store the dedicated index (~1Gb per million resources)
Easy to manage update and delete events	

#3 Browsing by Contributors using PostgreSQL index and Elasticsearch terms aggregation

This approach can be implemented in two steps:

1. Create new index for lowercase contributor values from instance jsons and browse by it
2. Retrieve counts using terms aggregation per contributor entity

Pros	Cons
It can be slightly better than option #1, because there is no need to store and manage dedicated index or table	Additional load to the existing data storage and mod-inventory-storage
No need to manage update and delete events	Additional index can slow down document indexing for mod-inventory-storage
	Slower than option #2

#4 Browse by range query and numeric representation of incoming value

This approach can be based on the Call-Number browsing. The main idea is to create a long value for the string and use it for the range query to limit the number of documents to aggregate.

Items to check/investigate:

- *Retrieve number of instances per each contributor*
- *Manage how to deal with redundant contributors in the result of terms aggregation (script query?)*

Pros	Cons
Approximately, the same performance as Call-Number Browsing.	Additional value must be stored within each document - numeric value for each contributor
No need to store dedicated Elasticsearch index, PostgreSQL table or index	There is no way to collect facets for contributor type/name
	Filter contributors by type will be hard too
	In case of large collisions (2000-3000 resources per contributr) - response will be slow