General goal

“Support to setup a cluster and some consultation in terms of importing the data (creating the right table structure) and writing a few queries to interrogate the data to be as fast as possible”

It's possible in general, but may require a powerful server to handle 1 billion documents. Handling 60 million documents shouldn't be a problem. All the tests below were conducted on a 1-billion-document dataset.

Data collection and ingestion

- “60 million docs now”
- “1 billion new docs each year”
- Data ingestion: “10-500 documents per second”
- “some of these documents repeat itself across many documents. (an example is that for the 60 million documents I have, unique texts are 2.2 million, giving us an average of around 30 identical texts each one in a different document). There are also some differences between the documents, but we don’t search or search by that (maybe just in extreme cases)”

Goal: “The indexer should update as much as possible in real time”. Ingestion rate: at least 500 docs per second.

Manticore provides true real-time data accessibility if you use a real-time index. Using this PHP script, it was proven that it's possible to load the sample data provided by Flaviu, multiplied millions of times, to Manticore in a real-time manner. With concurrency set to 10 and a batch size of 10,000, the average ingestion rate while loading 1 billion documents was 149K docs per second, which is much higher than 500.

```
snikolaev@dev2:~$ php load_flaviu.php 10000 10 100000 10000
preparing...
found in cache
querying...
finished inserting
Total time: 6724.8717930317
148702 docs per sec
```

After that, it took about 14 hours more to complete merging the disk chunks (using 1 core without read/write blocking):

```
[Fri Apr 28 06:25:05.240 2023] [1267203] rt: table flaviu: optimized
progressive chunk(s) 846 ( left 64 ) in 14h 23.8m
```

After that the table stats were:

```
root@dev2 ~ # echo "size per extension, GB"; for ext in spa spb spd spds spe sphi spi spidx spm spp spt ram; do echo -n ".$ext - "; sudo ls -la /var/lib/manticore/flaviu/*.$ext|awk '{sum += $5} END {print sum/1024/1024/1024}' ; done;
size per extension, GB
.spa -     34.0471 (row-wise plain attributes)
.spb -     8.14804 (blob attributes)
.spd -     68.0157 (doc list)
.spds -    147.952 (docstore)
.spe -     1.97206 (skiplists)
.sphi -    0.00132835 (attribute histograms)
.spi -     0.00404815 (dictionary)
.spidx -   22.4154 (secondary indexes)
.spm -     0.1164 (killed docs bitmap)
.spp -     9.90054 (hitlist)
.spt -     4.87426 (additional data structures to speed up lookups by document ids)
.ram -     0.0523361 (RAM chunk)
```

Search performance

- “The expected response time for a search and sort should be less than 0.2 - 0.3 seconds”. “A problem that I have with the current system is that for some words (for example “Cat”) in the db there are more than 4 million images containing this word. Because of this, the response time could be up to 5 seconds from the DB, which is unacceptable. This is one of the most important thing that I would like to fix beside the constant growth of the DB”

Goal: average response time < 0.2 - 0.3 seconds

To achieve the goal on the scale of 1 billion documents, it's important to:

- Make sure .spa (plain attributes), .spb (blob attributes), .spd (document list), and .spp (hit list) are locked in RAM to avoid high volume I/O spikes lowering the performance. You can estimate the required RAM using the command shown above. To make all the things mlocked, you can execute `alter table flaviu access_plain_attrs='mlock', access_blob_attrs='mlock', access_doclists='mlock' access_hitlists='mlock'`, or do the same when you create a table.
- Use the default row-wise storage for faster sorting by multiple attributes.
- Use powerful multi-core CPUs.
- Lower RAM consumption by implementing a well-thought-out schema using:
    - bit(N) data types for fields that don’t require full 32/64-bit integers

- stored-only fields when applicable.

On a commodity AMD Ryzen * 16 * 2 server, you can achieve the following performance on the 1-billion-document table:

```
mysql> select ids, pp, mi, mt, iw, ih, weight() from flaviu where match('anthropomorphic gamer cat boy furry fursona, hazel fur color, looking to the side, cute, trending on artstation, high quality, square profile picture') and mi > 0 and nv < 10 and iw = 512 and ih = 512 and ii = '' and im = '' and st = 1 order by weight() desc, pv desc, nv asc, dl desc, cd asc limit 400, 20 option ranker=expr('sum(doc_word_count + lcs)');
+--------------------------------------+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+------+------+------+------+----------+
|  ids                                                                                                                                                                                                       |  pp
| mi   | mt   | iw   | ih   | weight() |
+--------------------------------------+---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+------+------+------+------+----------+
| 00005d26-3ebe-4ba3-963e-fb016db03f71 | anthropomorphic gamer cat boy furry fursona, hazel fur color, looking to the side, cute, trending on artstation, high quality, square profile picture, digital art, smooth, sharp focus, vtuber, anime style |    3 | 2    |  512 |  512 | 44 |
| 00005d26-3ebe-4ba3-963e-fb016db03f71 | anthropomorphic gamer cat boy furry fursona, hazel fur color, looking to the side, cute, trending on artstation, high quality, square profile picture, digital art, smooth, sharp focus, vtuber, anime style |    3 | 2    |  512 |  512 | 44 |
…
20 rows in set (0.68 sec)

+----------------+-----------------+
| Variable_name  | Value           |
+----------------+-----------------+
| total          | 420             |
| total_found    | 1960000         |
| total_relation | eq              |
| time           | 0.676           |
| keyword[0]     | anthropomorphic |
…
```

Sorting the entire 1-billion-document data collection by the required four attributes alone doesn't take much longer:

```
mysql> select ids, pp, mi, mt, iw, ih, weight() from flaviu where match('') order by weight() desc, pv desc, nv asc, dl desc, cd asc limit 400, 20 option ranker=expr('sum(doc_word_count + lcs)'); show meta;
…
+----------------+------------+
| Variable_name  | Value      |
+----------------+------------+
| total          | 420        |
| total_found    | 1000000000 |
| total_relation | eq         |
| time           | 0.734      |
+----------------+------------+
4 rows in set (0.00 sec)
```

Using more powerful CPUs should let you achieve the 0.2-0.3 second goal on a single server and 1 billion documents.

Matching

- “A query has the following fields that I need to filter for each search:
    - pp: (string FTS)
    - np: (string FTS)
    - mi: (integer)
    - nv: (integer)
    - iw: (integer)
    - ih: (integer)
    - ii: (exact string) or empty
    - im: (exact string) or empty
    - st: (boolean)
    - There is an AND between these fields.”

Goal: pp, np, mi, nv, iw, ih, ii, im, st should be filterable.

Done. See the example query above.

Relevance

- “Sometimes the documents could contain parentheses, or commas, or other signs (“There could be () {} [] , . ? Maybe others too, but these are the most important” (these are important as it could define different image concepts, and ideal would be to take them in consideration)”
- “On the FTS side, what would be very useful for us is to know if is a full match or just a small partial match. If we could have a score something like 0-100% of the text of image we could decide if we return an already generated image (a score close to 100%) or generate a new image (a score closer to 0%). The score should be calculated as a phrase search, because the order of words is important”
- “the sorting should be done by the following fields:
    - fts score order descending for the field "pp”.
    - pv (integer) ordered descending.
    - nv (integer) order ascending.
    - dl (integer) order descending.
    - cd (integer) order ascending.”

Goals:

- **sort first by FTS weight, then by a set of attributes;**
    - No problem. See the query above.
- **account for special characters;**
- **account for words order.**

- You can use blend_chars and a custom ranker based on the word count and LCS (longest common sequence) (more tuning may be required, below is just for the sake of demonstration) like this:

```
mysql> drop table if exists t; create table t(f text) blend_chars='(,),{,},[,],.,?,U+2C'
expand_keywords='1'    blend_mode='trim_none,trim_head,trim_tail,trim_both,trim_all,skip_pure'
morphology='stem_en'; insert into t(f) values('cat, dog running on the ((bed))'),('cat, dog
running on the bed'),('cat dog running on the bed'),('cat dog running on the bed?'),('dog cat
running on the bed'); select *, weight() from t where match('(cat,|cat) dog run on the
(\\(\\(bed\\)\\)|bed)') option ranker=expr('sum(word_count + lcs)');
--------------
drop table if exists t
--------------

Query OK, 0 rows affected (0.03 sec)

--------------
create    table    t(f    text)    blend_chars='(,),{,},[,],.,?,U+2C'    expand_keywords='1'
blend_mode='trim_none,trim_head,trim_tail,trim_both,trim_all,skip_pure' morphology='stem_en'
--------------

Query OK, 0 rows affected (0.00 sec)

--------------
insert into t(f) values('cat, dog running on the ((bed))'),('cat, dog running on the
bed'),('cat dog running on the bed'),('cat dog running on the bed?'),('dog cat running on the
bed')
--------------

Query OK, 5 rows affected (0.01 sec)

--------------
select *, weight() from t where match('(cat,|cat) dog run on the (\\(\\(bed\\)\\)|bed)')
option ranker=expr('sum(word_count + lcs)')
--------------

+---------------------+---------------------------------+----------+
| id                  | f                               | weight() |
+---------------------+---------------------------------+----------+
| 5838779201779400724 | cat, dog running on the ((bed)) |       14 |
| 5838779201779400725 | cat, dog running on the bed     |       12 |
| 5838779201779400726 | cat dog running on the bed      |       11 |
| 5838779201779400727 | cat dog running on the bed?     |       11 |
| 5838779201779400728 | dog cat running on the bed      |        9 |
+---------------------+---------------------------------+----------+
5 rows in set (0.00 sec)
```

- Matching fullness
  - Since you have to use a custom ranker and not just `ranker=wordcount`, which would be helpful in this case, the next easiest thing you can probably do is to use highlight() and then calculate the matching fullness in the app based on the number of highlighted words:

```
select highlight(), weight() from t where match('(cat,|cat) dog run on the
(\\(\\(bed\\)\\)|bed)') option ranker=expr('sum(word_count + lcs)')
--------------

+---------------------------------------+----------+
| highlight()                           | weight() |
+---------------------------------------+----------+
| <b>cat, dog running on the ((bed))</b> |       14 |
```

```
| <b>cat, dog running on the bed</b>     |       12 |
| <b>cat dog running on the bed</b>      |       11 |
| <b>cat dog running on the bed?</b>     |       11 |
| <b>dog cat running on the bed</b>      |        9 |
+----------------------------------------+----------+
5 rows in set (0.00 sec)
```

- You may also want to use packedfactors() since it's easier, but be aware that it will affect the search performance, which is not what you want.

Response

- "The result should return the
  - "id",
  - "pp",
  - "mi",
  - "mt",
  - "iw",
  - "lh",
  - "id" fields."
- "The results should be paginated by offset + limit". "Probably max to 300-400 results"

Not a problem. See the query above.