

# Search Pipeline use cases

Below is a list of search pipelines enabling you to alternate the results of a search process based on the desired capability. You can use COMPASS 'new pipeline from text' functionality to copy and paste pipeline code and see the results.

The Bury, Boost, Pin, Hide all uses the default dynamic index construct of the form:

```
{
  "mappings": {
    "dynamic": true
  }
}
```

The last Pipeline - diacritics abstraction - uses a custom analyser definition.

## Bury

Bury pipeline function can be achieved by matching the desired documents you want to bury and override the Lucine relevancy score with a constant value. Application logic leverages this constant score to process documents. Hence all matching documents will yield a uniform score effectively burring these documents amongst the entire dataset.

The screenshot displays the MongoDB Compass interface with three search pipelines configured for the 'amazonEcommMaster' database. The left sidebar shows the database structure with collections like 'customers', 'products', and 'local'. The main panel shows the 'Aggregations' tab with three stages:

- \$search**: The search query is `{ "query": "Longboards", "path": "Product Name" }`. The output shows 9 documents, including 'Rayne Longboards Demosnead Longboard Complete' and 'Rayne Longboards Minotaur 34" Double Kick Skateboard'.
- \$match**: The match query is `{ "selling Price": { "$lt": 70 } }`. The output shows 7 documents, including 'Rayne Longboards Demosnead Longboard Complete' and 'Rayne Longboards Minotaur 34" Double Kick Skateboard'.
- \$project**: The project query is `{ "Product Name": 1, "selling Price": 1, "buried_score": { "$literal": 1 } }`. The output shows 7 documents, including 'Rayne Longboards Demosnead Longboard Complete' and 'Rayne Longboards Minotaur 34" Double Kick Skateboard'.

## Boost

The boost function enables you to augment the original Lucine relevance score enabling you to influence the overall score by using additional factors. For instance within the context of an Ecommerce site, the ability to control the order of presentation of various articles for sales (ie documents) can be a function of the native relevance score from Lucine multiplied by the number of likes associated with each product items.

The screenshot displays the MongoDB Compass interface for a database named 'hack2022.amazonEcommMaster'. The 'Aggregations' tab is active, showing a pipeline with two stages: '\$search' and '\$project'.

**\$search stage:** The query is `{ $text: { $query: 'Inline Skate', $score: { function: { multiply: [ { path: '$likes', $ifNull: { $literal: 1, as: 'likes' } } ] } } } }`. The output shows a sample of 20 documents, each with a 'score' field representing the relevance score.

**\$project stage:** The query is `{ $project: { _id: 0, 'Product Name': 1, 'score': { $meta: 'searchScore' } } }`. The output shows a sample of 20 documents, each with a 'Product Name' and a 'score' field.

## Pin

Pinning of documents can be implemented by controlling the quantity of documents returned by the pipeline. This is easily done by defining a limit stage within your aggregation pipeline. The Pin function can be utilized in addition to other functions - like boost. You can control the number of documents to Pin just by changing the limit criteria - easy...

The screenshot displays the MongoDB Compass interface for an aggregation pipeline named 'hack2022.amazonEcommMaster'. The pipeline consists of three stages: Search, Limit, and Project.

- Search Stage:** The search query is defined as:
 

```
1 * {
2   text: {
3     path: 'Product Name',
4     query: '"Inline Skate"',
5     score: {
6       function: {
7         multiplies: [
8           {
9             path:
10              value: 'likes',
11              undefined: 1
12            },
13          ],
14          score: 'relevance'
15        }
16      }
17    }
18  }
19 }
20 }
21 }
```

 The output shows a sample of 20 documents, each containing product details like \_id, Uniq Id, Product Name, Category, About Product, Product Specification, Technical Details, Product Url, quantity, and selling Price.
- Limit Stage:** The limit is set to 1 document. The output shows a single document with the same structure as the Search stage output.
- Project Stage:** The project stage is defined as:
 

```
1 * {
2   _id: $_,
3   "Product Name": 1,
4   score: {
5     $meta: 'searchScore'
6   }
7 }
```

 The output shows a single document with the \_id, Product Name, and a calculated score.

## Hide

Hide capability can be implemented by using a compound search function. In the pipeline below we are searching for products that are made of Titanium and we are excluding products that are made in bamboo or Fiberglass. This can be accomplished using a single search stage within Aggregation Pipeline, There is no need to add any additional attributes or business logic to implement this function.

The screenshot shows the MongoDB Compass interface with the Aggregations tab selected. The pipeline is as follows:

```

1 {
2   compound: {
3     must: [
4       {
5         text: {
6           query: 'Titanium',
7           path: 'Product Name'
8         }
9       },
10    ],
11    mustNot: [
12      {
13        text: {
14          query: 'Bamboo Fiberglass',
15          path: 'Product Name'
16        }
17      }
18    ]
19  }
20 }

```

The output after the \$search stage shows three documents:

```

{ "_id": "ObjectID('623098252f29c729af25fc6f')", "Uniq Id": "3171a01da9494ac5e4d63e9bdfaf04", "Product Name": "IELLO Titanium Wars Board Game", "Category": "Toys & Games | Games & Accessories | Board G", "About Product": "Make sure this fits by entering your mo", "Product Specification": "ProductDimensions:6x6x1.5Inches", "Technical Details": "show up to 2 reviews by default As", "Product Url": "https://www.amazon.com/IELLO-Titanium-War", "quantity": 1, "selling Price": 15.99 }
{ "_id": "ObjectID('623098232f29c729af25ede1')", "Uniq Id": "47bd8941a3f713a3b0b05fb8b8dda610", "Product Name": "Westcott Titanium Bonded Razor Paper Cut", "Category": "Office Products | Office & School Supplies |", "About Product": "Make sure this fits by entering your mo", "Product Specification": "ProductDimensions:8x1x3.2Inches", "Technical Details": "Size:One Pack The Westcott Titanium", "Product Url": "https://www.amazon.com/Westcott-Titanium-" }
{ "_id": "ObjectID('62317c4c9b61dbj)", "Uniq Id": "4c9b61dbj)", "Product Name": "DB Lc", "Category": "Sports & Skateboar", "About Product": "Make", "Product Specification": "ProductDimensions:8x1x3.2Inches", "Technical Details": "Size:One Pack The Westcott Titanium", "Product Url": "https://www.amazon.com/Westcott-Titanium-" }

```

The output after the \$project stage shows the same three documents with a score field added:

```

{ "_id": "ObjectID('623098252f29c729af25fc6f')", "Product Name": "IELLO Titanium Wars Board Game", "score": 4.112264633178711 }
{ "_id": "ObjectID('623098232f29c729af25ede1')", "Product Name": "Westcott Titanium Bonded Razor Paper Cut", "score": 3.587188482284546 }
{ "_id": "ObjectID('62317c4c9b61dbj)", "Product Name": "DB Lc", "score": 3.587188482284546 }

```

## Diacritics Abstraction

This capability applies to ECommerce websites that service global customers, where various locals use diacritic symbols. For instance in French a word can be spelled out correctly using various representations ( for instance the first name benoit can also be spelled benoît). The custom analyser will deliver documents which contain both representations of benoit (and benoît) without any regards to the actual encoding.

By implementing a custom analyser we can abstract the diacritic representation, delivering a superior user experience.

Using Mongo shell let's query the amazonEcommMaster

Using the default analyser - noticed that the query only returns a single document with the author name = benoit

```

MongoDB Enterprise atlas-3ej8tw-shard-0:PRIMARY> db.amazonEcommMaster.aggregate([
...   {
...     $search: {
...       "index": "default",
...       "text": {
...         "query": "benoît",
...         "path": "author"
...       }
...     }
...   },
...   {
...     $project: {
...       "_id": 1,
...       "author": 1
...     }
...   }
... ])
{ "_id" : ObjectId("6230e6a892cdccde2e4a549f"), "author" : "benoît" }
MongoDB Enterprise atlas-3ej8tw-shard-0:PRIMARY>

```

Using the custom analyser - asciiConverter, noticed the engine returns both representations benoit (and benoît) regardless of the search query parameter thus abstracting the diacritic representation of the attribute author.

```

... }]]
MongoDB Enterprise atlas-3ej8tw-shard-0:PRIMARY> db.amazonEcommMaster.aggregate([
...   {
...     $search: {
...       "index": "asciiConverter",
...       "text": {
...         "query": "benoît",
...         "path": "author"
...       }
...     }
...   },
...   {
...     $project: {
...       "_id": 1,
...       "author": 1
...     }
...   }
... ])
{ "_id" : ObjectId("6230e67192cdccde2e4a549e"), "author" : "benoit" }
{ "_id" : ObjectId("6230e6a892cdccde2e4a549f"), "author" : "benoît" }
MongoDB Enterprise atlas-3ej8tw-shard-0:PRIMARY>

```

Hope you enjoy