Data Modelling

Sancheeta Kaushal | Ex-EM Blinkit

Disclaimer

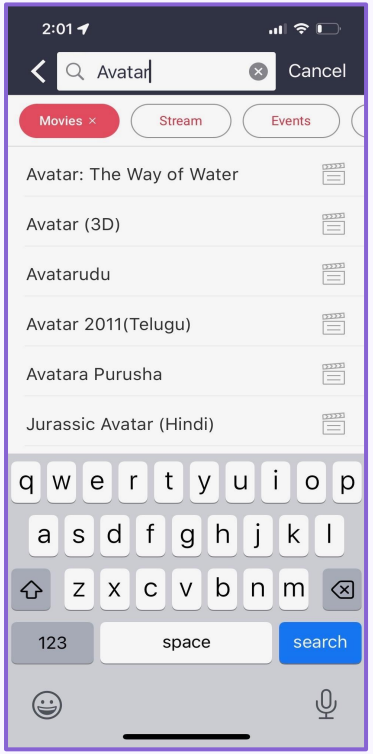
1. What makes learning on Airtribe different than learning from Youtube ?
2. Community - How many of you leverage it ?
3. Teaching methods like breakout rooms - How many of you engage when put in such situations ?
4. Skills - Are you still thinking of just getting a job or you want to become a great engineer ? Are you practising things to deepen your understanding ?

Problem Statement

Problem Solving Case: Bookmyshow

Continuing with our Bookmyshow case study.

1. Implement code for quick retrieval of static movie data - Name, Cast, Crew, Movie Plot, Runtime, Language, Genre, etc.
2. Build an endpoint for storing and retrieval of comments and ratings for a given movie. Which database would you choose and why ?
3. Build an endpoint where one can search for movies and theaters. Refer image on the right.

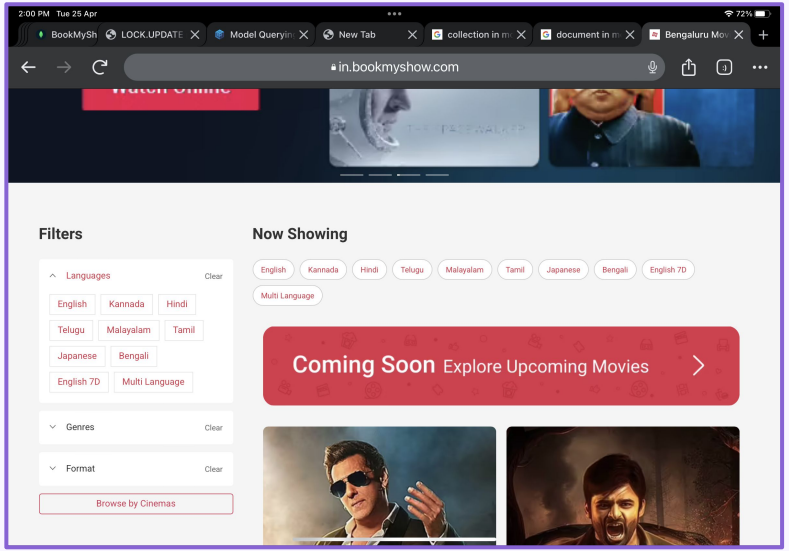


Airtribe

Problem Statement

Problem Solving Case: Bookmyshow

4. Implement language (Hindi, English, Kannada, Telgu etc.) and genre (Romantic, Action, Scifi) and format (2D, 3D, IMAX) filters.



Airtribe

What all skills you need to learn to build this ?

- Factors to consider when deciding on a database
- Learn different types of databases that exist
- Understand what type of database should be used in what scenario
- Understanding document databases and related concepts in depth

Agenda

- Document Databases - MongoDB
- Document based Search Databases - Elasticsearch

Mongo DB

- Distributed document based database
- Uses Binary JSON format (BSON).
    - Adds data types and binary encoding.
    - Additional data types - binary data, date-time values, regular expressions
    - Binary representation so more efficient in terms of storage and data transfer as it’s faster to serialise and deserialise due to encoding.
- Content management systems, E-commerce reviews, IoT analytics, Logging
- Data is stored as collections each containing multiple documents

Mongo DB

RDBMS	MongoDB
Tables	Collections
Rows	Documents

```json
{
  "name": "John Doe",
  "age": 30,
  "email": "john.doe@example.com",
  "isSubscribed": true,
  "interests": ["sports", "movies", "reading"]
}
```

```
\x1E\x00\x00\x00            // Total BSON document size (in bytes)
\x02                        // String type (key type)
name\x00                    // Key "name" (null-terminated)
\x0A\x00\x00\x00John Doe\x00  // Value "John Doe" (null-terminated string)
\x10                        // 32-bit integer type (key type)
age\x00                     // Key "age" (null-terminated)
\x1E\x00\x00\x00            // Value 30 (32-bit integer, little-endian)
\x08                        // Boolean type (key type)
isSubscribed\x00            // Key "isSubscribed" (null-terminated)
\x01\x00                    // Value true (boolean, 0x01 represents true)
\x04                        // Array type (key type)
interests\x00               // Key "interests" (null-terminated)
\x16\x00\x00\x00            // Array size (in bytes, 22 bytes)
\x02\x00\x00\x00sports\x00  // Array element "sports" (null-terminated stri
\x02\x00\x00\x00movies\x00  // Array element "movies" (null-terminated stri
\x02\x00\x00\x00reading\x00 // Array element "reading" (null-terminated str
\x00                        // End of document (null terminator)
```

Airtribe

Setting up Mongo in Codebase

- Create a new database deployment on Mongodb Atlast cloud.
- Now you need to install Mongodb and Mongoose (ODM) Libraries in package.json as dependencies.
- Next we create a config to write code to connect with Cloud Mongo.
- We test the connection on the server launch.
- Install mongosh to run shell from local environment.
- What to do when you can’t find ODM/ORM that connects to a database ? - Link

Nested data structure in Mongo.

Now there are no joins in Mongo, so how do we get nested data ?

Projections, Embeddings & Aggregates

Mongo DB

Projections in MongoDB refer to the ability to specify which fields or properties of a document should be returned in the query results. Much like select a, b, c in SQL.

Embedding is a practice of storing related data within a single document by embedding one document inside another.

Mongod Syntax

```
Projections
db.inventory.find({ "status": "processed"}, { "item": 1, "status": 1, "_id": 0 } )

Embedding
db.user.find({"posts.comment_id":  {$eq : 350}})
```

Airtribe

Example – Projections

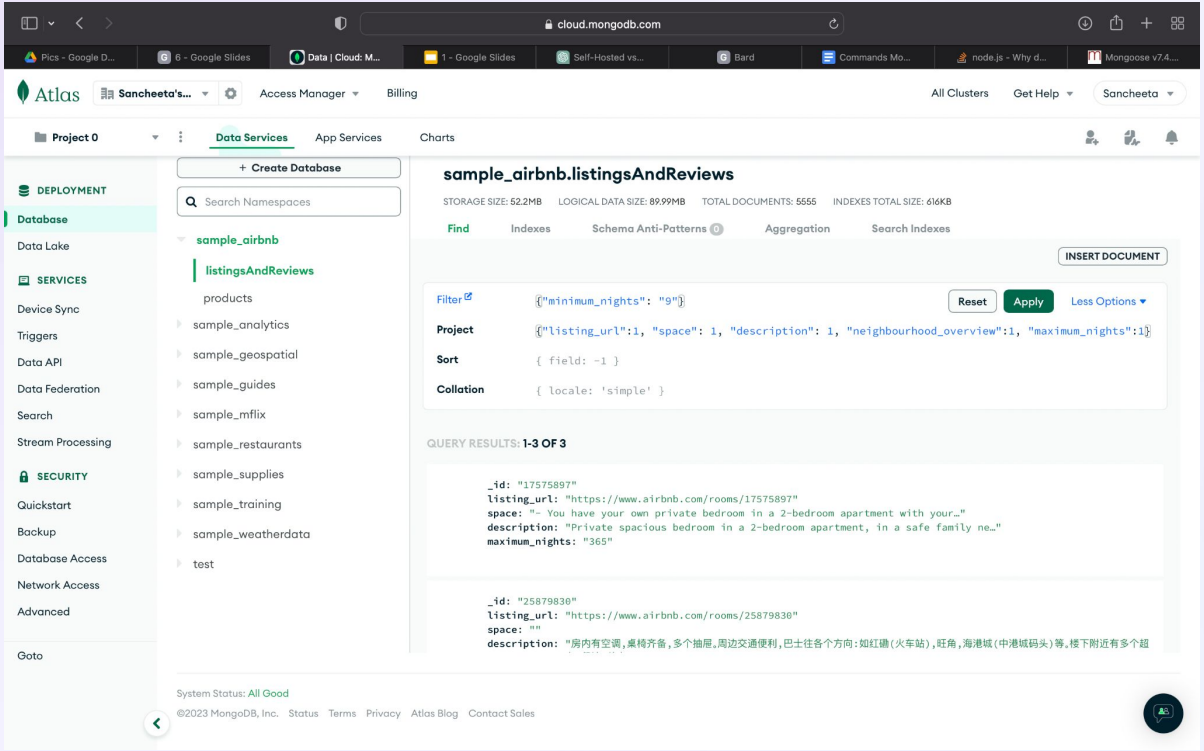
Go to Airbnb collection and find the properties where minimum stay is 9 days. How many documents are returned. Only select the following columns

- Listing URL
- Space
- Description
- Neighbourhood Overview
- Maximum Nights

Write the Query to get this data from Atlas.

Airtribe

Example – Projections



Airtribe

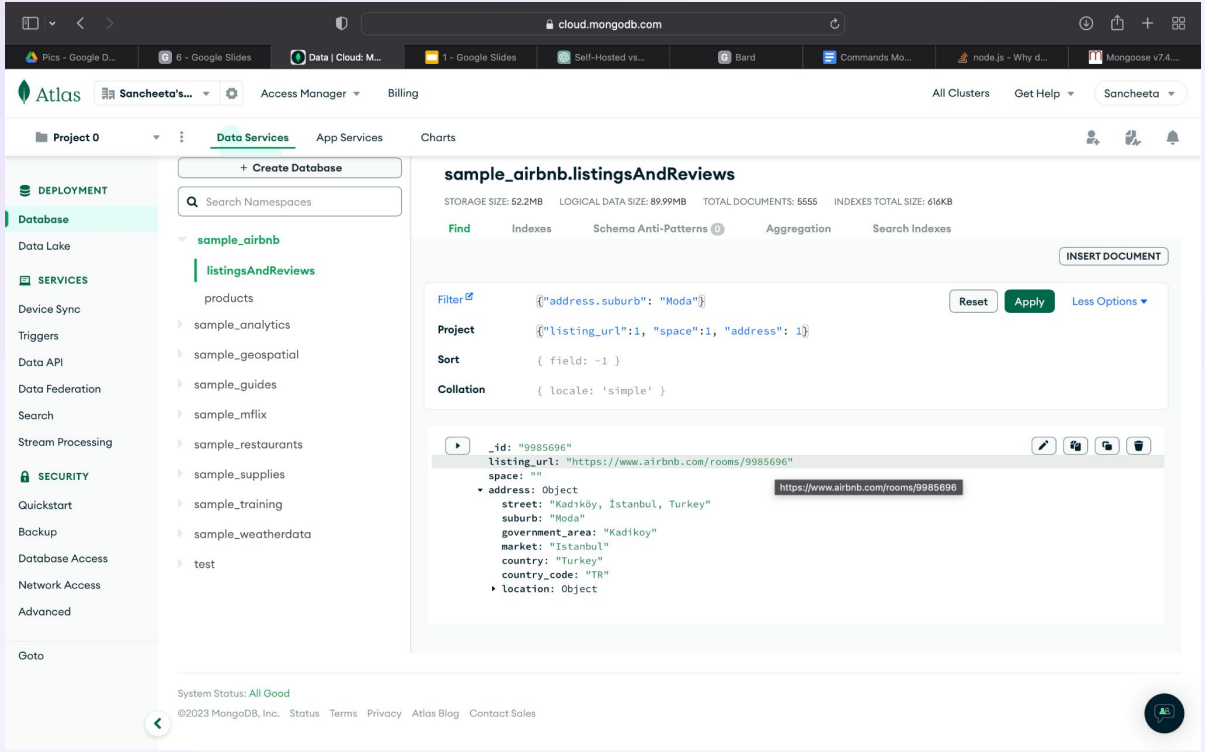
Example – Embeddings

Go to Airbnb collection and find the properties whose address has a suburb location of Moda in Turkey. Only select the following columns

- Listing URL
- Space
- Address

Write the Query to get this data from Atlas.

Example – Embeddings



Airtribe

Mongo DB – Availability

- Replica Sets - Multiple MongoDB nodes, where one node acts as the primary and others act as secondary replicas
- The primary replica handles all write operations and replicates data to the secondary replicas
- MongoDB's replica set members continuously monitor each other's status through heartbeat messages.
- In case of primary replica failure, the node misses a heartbeat and other nodes initiate an election process to elect a new primary replica.

Mongo DB – Consistency

- ACID Compliant transactions
- Eventual consistency model
- Asynchronous propagation of data across primary and replicas
- Data replicas are temporarily inconsistent with each other but will eventually converge to a consistent state without any further updates.
- Reading data immediately after writing can lead to inconsistent results
- Versioning, conflict resolution or business logic to handle inconsistencies.

Mongo DB – Consistency

- Ways to ensure consistency
  - Write concern specifies the number of replicas or nodes that must acknowledge a write operation before it is confirmed.
  - Read Preference allows you to specify the preferred replicas to read from based on factors such as latency, availability, and consistency requirements. Eg. Primary, Secondary, Primary Preferred, Secondary Preferred, Nearest
  - Read Concern determines how reads are performed in relation to the most recent write operations. Eg. Local, Majority & Linearlizable

Mongo DB – Consistency

- Local Read Concern
  - Read operation may return the most recent data from a write operation, even if data has not been fully replicated across all replica set members.
- Majority Read Concern
  - Reads are guaranteed to reflect the majority of data from the replica set, ensuring strong consistency.
- Linearlizable Read Concern
  - Reads reflect the most recent write operation or an error if a write is in progress.

Mongo DB – Scalability

- Sharding is a technique used to horizontally partition data across multiple shards in a distributed MongoDB. Can only be done via mongos instance.
- Shards are individual servers (or a replica set) that collectively store and manage a large dataset.
- Shard key is a field or set of fields chosen from the documents in a collection that determines how to distribute data across shards.
- Config Server is a server (or a replica set) that stores metadata about the sharded data. They route queries to the appropriate shards and maintain the overall cluster's configuration.

Mongo DB – Scalability

Query router (mongos) is a component responsible for

- Evaluating the shard key
- Routing the client queries to the appropriate shards in a sharded cluster.
- Collecting and merging query results from multiple shards.
- It is also responsible for connection pooling.

Multiple query routers (mongos instances) can be deployed in the cluster for high availability and scalability.

Creating a config server

Mongod Syntax

```
mongod --serverType --dbpath /opt/homebrew/var/mongodb/db_path --port port
--fork --logpath log.filename --replSet replica_set
```

Mongo Config Server

```
mongod --configsvr --dbpath /opt/homebrew/var/mongodb/cfg0 --port 26050 --fork
--logpath log.cfg0 --replSet cfg

mongod --configsvr --dbpath /opt/homebrew/var/mongodb/cfg1 --port 26051 --fork
--logpath log.cfg1 --replSet cfg

mongod --configsvr --dbpath /opt/homebrew/var/mongodb/cfg2 --port 26052 --fork
--logpath log.cfg2 --replSet cfg
```

Creating a Mongosh

Mongosh Syntax

```
mongosh --host localhost --port 26050

> rs.initiate()  //Create a replicaset

> rs.add('localhost:26301')    //Add servers to replicaset

> rs.add('localhost:26302')

> rs.status()
```

Creating a shard server replica set

Mongod Shard Server

```
mongod --shardsvr --replSet a --dbpath /opt/homebrew/var/mongodb/a0 --port 26000 --fork --logpath log.a0

mongod --shardsvr --replSet a --dbpath /opt/homebrew/var/mongodb/a1 --port 26001 --fork --logpath log.a1

mongod --shardsvr --replSet a --dbpath /opt/homebrew/var/mongodb/a2 --port 26002 --fork --logpath log.a2

//Add the replicasets via mongosh
```

Creating a query router server

Mongos Query Router

```
mongos --configdb
"cfg/localhost:26050,localhost:26051,localhost:26052" --fork --logpath
log.mongos1 --port 26061

mongos --configdb
"cfg/localhost:26050,localhost:26051,localhost:26052" --fork --logpath
log.mongos2 --port 26062
```

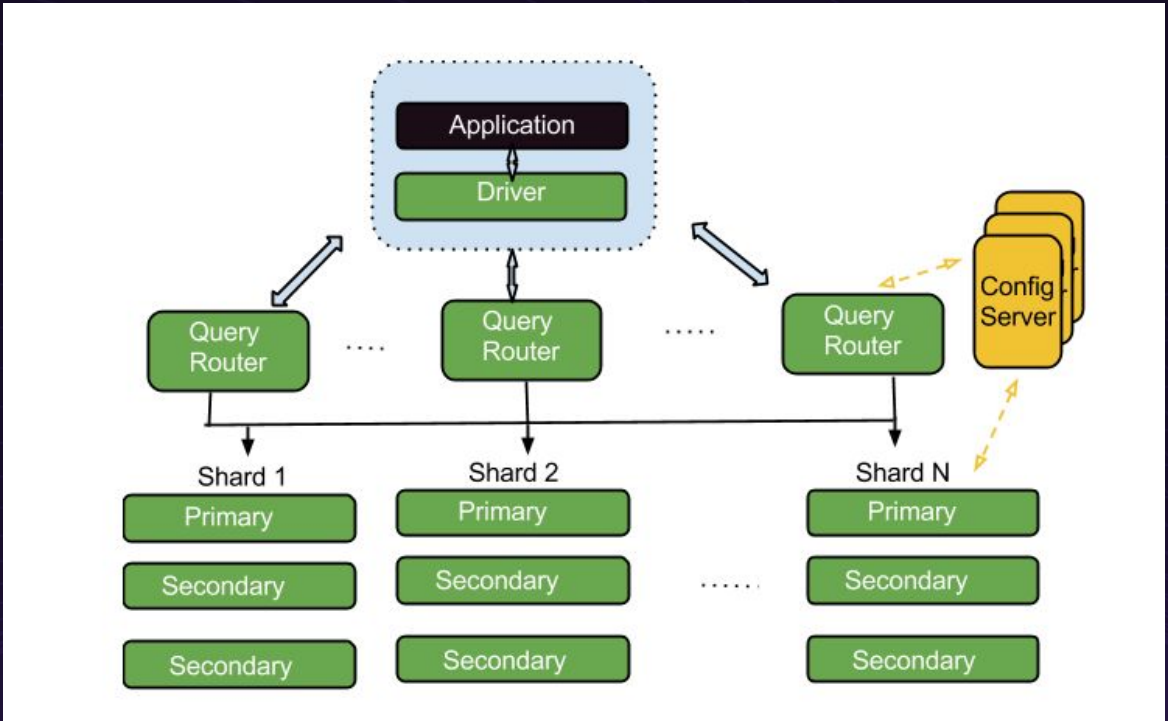
Enabling sharding

Mongos Query Router

```
> mongosh –host localhost --port 26061 //mongos server

> sh.addShard(a/localhost:26000)  //Add primary database URL

> sh.status()

> sh.enableSharding("db_name");

> sh.shardCollection("db_name.collection", {_id: 1}) // Asc order by id

> use config;

> db.shards.find()
```

Airtribe

Mongo Server Interactions



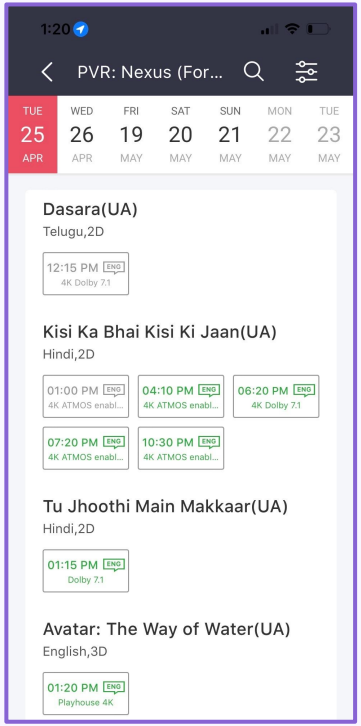
Airtribe

Limitations of MongoDB

Problem Solving

Example Case Demo:

- Let’s model this page using MongoDB instead of an RDBMS.
- Evaluate this decision based on factors for deciding a database.
- Define the collections and documents.
- Do you think we can model it as one collection ? What benefits or issues do you see ? Think if there are more reads than writes or vice versa.



Airtribe

Why do we need a document database specifically for search?

Search is a complex problem.
ES solves all the use cases for search scenarios.

Airtribe

Elasticsearch – Document database

- Distributed search and analytics engine built on top of Apache Lucene, a powerful search library optimized for fast indexing and searching of textual data.
- Data stored in a JSON-like format.
- Never use as a primary database. Always use as an index.
- Full Text Search, Log analysis and GIS
- High performance due to inverted index structure.

Inverted Index

```
{
  "texts": [
    "I like to eat apples.",
    "Apples are delicious.",
    "I prefer oranges over apples."
  ]
}
```

Term	Documents
I	1, 3
like	1
to	1
eat	1
apples	1, 2, 3
are	2
delicious	2
prefer	3
oranges	3
over	3

Airtribe

Elasticsearch – Document database

- Document is a basic unit of information in Elasticsearch.
- An index is a logical namespace that holds one or more documents.
- A mapping defines the schema or structure of the documents in an index. It specifies the fields and their data types. _doc is the only valid mapping type now.
- Different field types - text, keyword, integer, date, boolean, float, double, geo-point
- Sharding and Replication by default.
- An index can hold only a single type of document now.

Elasticsearch – Mapping

- The "properties" section within the mapping defines the fields of the index.
- The "title" field is of type "text" and uses the "standard" analyzer, tokenizes the text and converts it to lowercase.
- The "author" field is of type "keyword," meaning it will not be analyzed and can be used for exact match queries.
- The "publication_date" field is of type "date," which allows Elasticsearch to handle date-related queries efficiently.
- The "description" field is of type "text" and uses the "english" analyzer, which performs stemming and stopword removal to improve text search relevance.

```
PUT /books
{
  "mappings": {
    "properties": {
      "title": {
        "type": "text",
        "analyzer": "standard"
      },
      "author": {
        "type": "keyword"
      },
      "publication_date": {
        "type": "date"
      },
      "description": {
        "type": "text",
        "analyzer": "english"
      }
    }
  }
}
```

Airtribe

Setting up Elasticsearch in Codebase

- Download elasticsearch from the official website and install it on your local.
- Now you need to install elasticsearch library in package.json as dependencies.
- Next we create a config to write code to connect with ES running on port 9200.
- We test the connection on the server launch. We check if the status is green. If yes, we are good to go. If yellow, replica shards are missing, so not fault tolerant.
- Let's now create index, get index , bulk index data and get index data.
- Elasticsearch REST API

Elasticsearch – DSL & Query Context

- DSL (Domain-Specific Language) refers to the query and filter syntax to perform operations like searching, filtering, aggregating data within an index.
- DSL follows a JSON-based syntax.
- DSL components: Query Context, Filter Context, Aggregations.
- Query context is used for determining document relevance using scores and ranks based on how well they match the search criteria.
- Eg. full-text search queries, phrase queries, and fuzzy queries.
- For more context, refer the official doc link.

Airtribe

Elasticsearch – Query Context Basics

- "match": Used to perform full-text search on text fields. It analyzes the query text and returns documents that match the terms in the query.
- "match_phrase": Used to find documents containing an exact phrase in a specific field. Returns documents where the terms appear in the specified order.
- "term": Used for exact matching of terms in keyword fields. It does not analyze the query text, making it suitable for keyword fields or fields with exact values.
- "terms": Used to match documents that have one or more exact values in a keyword field. It allows multiple values for a single field in a single query.

Elasticsearch – Query Context Basics

- "range": Used to match documents within a specified range of values. It is commonly used with numeric or date fields.
- "exists": Used to find documents that have a specific field in them, regardless of its value.
- "prefix": Used to find documents where a field starts with a specific prefix.
- "wildcard": Used to perform wildcard searches on keyword fields. It supports using wildcards like "*" and "?" in the search term. Eg *nut*, *n?t*
    - Peanut, Notebook, Coconut, Next

Elasticsearch – Query Context Basics

- "regexp": Use regular expressions to perform pattern-based searches on keyword fields.
- "bool": The "bool" query is a powerful compound query that allows you to combine multiple queries using boolean logic (e.g., "must," "should," "must_not").
- "multi_match": Allows you to search multiple fields simultaneously and return documents that match any of the specified fields.
- "query_string": Used to perform more complex searches using a query string with support for Boolean operators, wildcards, and other features.

Elasticsearch – Query Context

Queries in the query context are used in the "query" section of Elasticsearch queries.

```
{
  "query": {
    "bool": {
      "must": [
        { "term": { "category": "electronics" }},
        { "range": { "price": { "gte": 100, "lte": 500 }}}
      ],
      "must_not": [
        { "term": { "brand": "Apple" }}
      ],
      "should": [
        { "term": { "color": "red" }},
        { "term": { "size": "large" }}
      ]
    }
  }
}
```

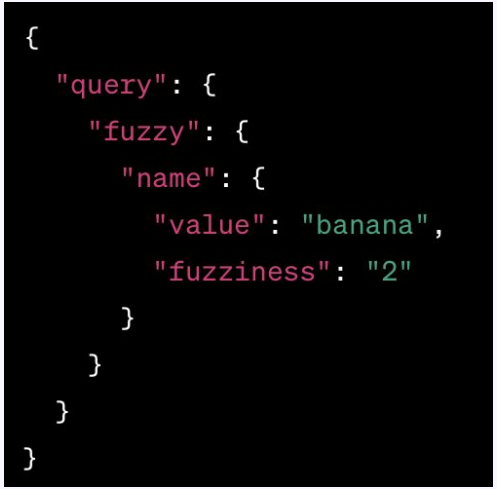
```
{
  "query": {
    "bool": {
      "must": [
        { "match": { "title": "apple" }}
      ],
      "should": [
        { "match_all": {}},
        { "match_phrase": { "description": "high-quality product" }}
      ]
    }
  }
}
```

Airtribe

Elasticsearch – Fuzziness Operations

- Substitutions: Replacing one character with another.
- Insertions: Adding an extra character in the term.
- Deletions: Removing a character from the term.
- Transpositions: Swapping adjacent characters.

If the original term is "banana" and the fuzziness is set to 2, the fuzzy query matches - "banana" (exact match), "bananas" (insertion of 's'), "panana" (substitution of 'p' for 'b'), "bonana" (substitution of 'o' for 'a')



```
{
  "query": {
    "fuzzy": {
      "name": {
        "value": "banana",
        "fuzziness": "2"
      }
    }
  }
}
```

Airtribe

Elasticsearch – Query Context

A search query for finding documents containing the term "apple" in the "title" field.

```
{
  "query": {
    "match": {
      "title": "apple"
    }
  }
}
```


```
{
  "hits": {
    "total": 3,
    "hits": [
      {
        "_id": "1",
        "_score": 0.45329493,
        "_source": {
          "title": "Apple iPhone",
          "description": "The latest Apple iPhone model"
        }
      },
      {
        "_id": "2",
        "_score": 0.2579254,
        "_source": {
          "title": "Apple MacBook Pro",
          "description": "Powerful laptop by Apple"
        }
      },
      {
        "_id": "3",
        "_score": 0.16987906,
        "_source": {
          "title": "Samsung Galaxy",
          "description": "A competitor to Apple devices"
        }
      }
    ]
  }
}
```


Airtribe

Elasticsearch – Filter Context

- Filter context is used for narrowing the search space by exact matching and filtering → reduce the output set.
- Filtered queries in the filter context are used to match documents that meet certain conditions without affecting the relevance score.
- They are often used for exact matches, range queries, term queries, and other non-scoring conditions.
- They can significantly improve search performance by reducing the number of documents to be scored.

Airtribe

Elasticsearch – Filter Context

Filter context queries can be used in the "filter" section of Elasticsearch queries.

```
{
  "query": {
    "bool": {
      "filter": [
        { "term": { "category": "electronics" }},
        { "range": { "price": { "gte": 100, "lte": 500 }}}
      ]
    }
  }
}
```

Airtribe

Elasticsearch – Filter Context

Consider a filter query for finding documents with a "price" range between 100 and 500.

```
{
  "query": {
    "bool": {
      "filter": [
        { "range": { "price": { "gte": 100, "lte": 500 }}}
      ]
    }
  }
}
```


```
{
  "hits": {
    "total": 2,
    "hits": [
      {
        "_id": "1",
        "_score": 0,
        "_source": {
          "title": "Apple iPhone",
          "price": 399
        }
      },
      {
        "_id": "2",
        "_score": 0,
        "_source": {
          "title": "Samsung Galaxy",
          "price": 299
        }
      }
    ]
  }
}
```


Airtribe

Limitations of Elasticsearch

- Learning curve as querying and aggregations can be complex.
- ES requires a significant amount of resources, especially as the data size and query complexity increase.
- Understanding the architecture, data modeling, query optimization, and best practices may require some time.
- Upgrades may need careful planning as changes are non-trivial in nature.
- High-speed indexing of large volumes of data may require careful optimization and tuning.

Elasticsearch – Filter Context

What's the scenario where I have to use query context as filter context can't be used ?

An e-commerce website wants to implement a search functionality that retrieves products matching certain criteria but also takes into account the popularity of the products. The popularity is determined by the number of sales and customer ratings.

```
{
  "query": {
    "bool": {
      "must": [
        { "match": { "title": "electronics" }}
      ],
      "filter": [
        { "range": { "price": { "gte": 100, "lte": 500 }}}
      ]
    }
  },
  "sort": [
    { "sales": { "order": "desc" }},
    { "rating": { "order": "desc" }}
  ]
}
```

Airtribe

Thank You!

Airtribe