# Comprehensive Guide to Elasticsearch Data Types

## 1. Introduction to Elasticsearch Data Types

Before we get into mapping, let’s talk about data types.

Elasticsearch supports a wide variety of data types to cater to different use cases. These data types define how data is stored, indexed, and queried.

Basic data types include common ones like integers, floats, strings, and booleans. **Specialized data types**, such as `ip` for IP addresses or geospatial types for location-based queries, extend the functionality of Elasticsearch.   
Some data types to support **Auto-Completion**.  
This document explores these data types and their applications.

## 2. Basic Data Types

Elasticsearch provides several basic data types that are analogous to those found in programming languages. Examples include:  
- `integer`, `long`, `short`, `byte`: Numeric types for whole numbers.  
- `float`, `double`: Types for decimal numbers.  
- `boolean`: For true/false values.  
- `date`: To store date values.

Apart from these, Elasticsearch also supports specialized data types like `ip` for IP addresses, and others designed for features like auto-completion and geospatial searches.

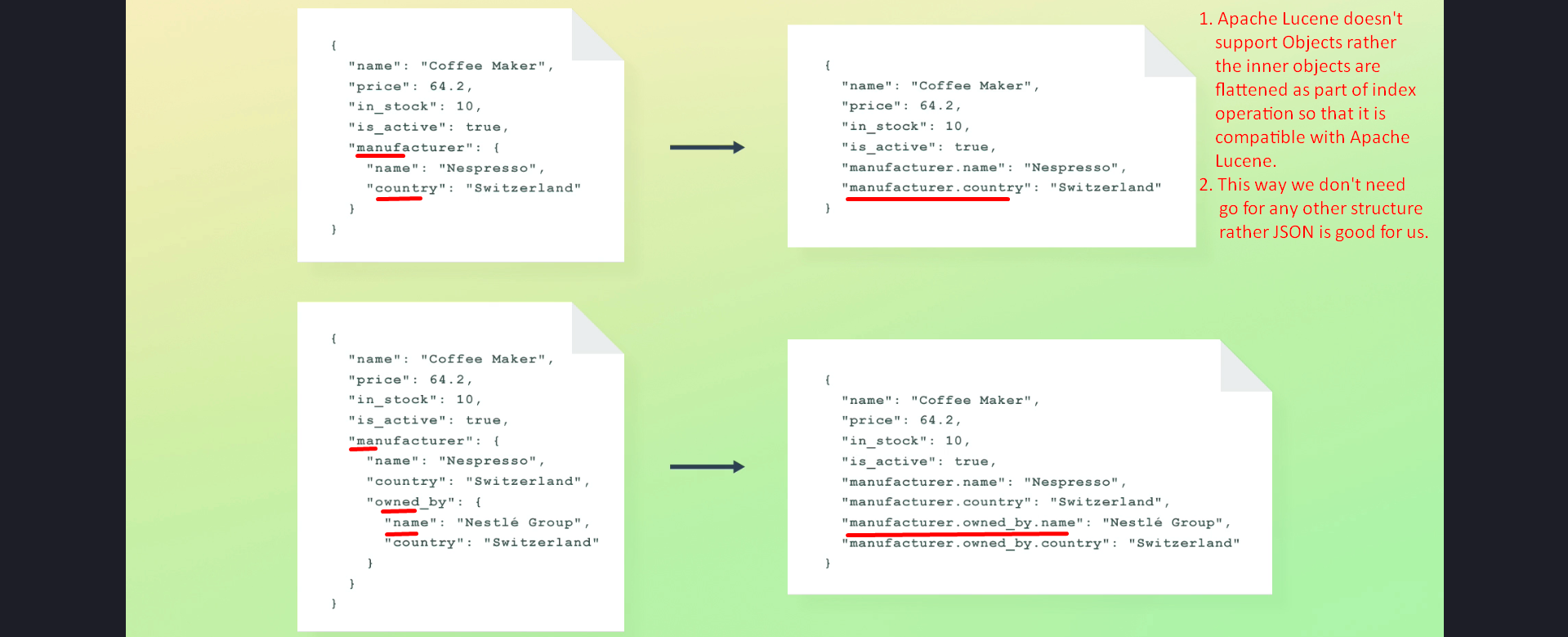
## 3. Complex Data Types - Object

1. The `object` data type represents JSON objects, which are fundamental to Elasticsearch.

2. Documents in Elasticsearch are JSON objects and can contain nested objects to form a hierarchy.

3. Mapping for objects uses a `**properties**` key to define fields and their types, rather than a `type` key.

4. Example: **Below is the document and its mapping.**  
 **5. Example: Mapping for Inner Object.**

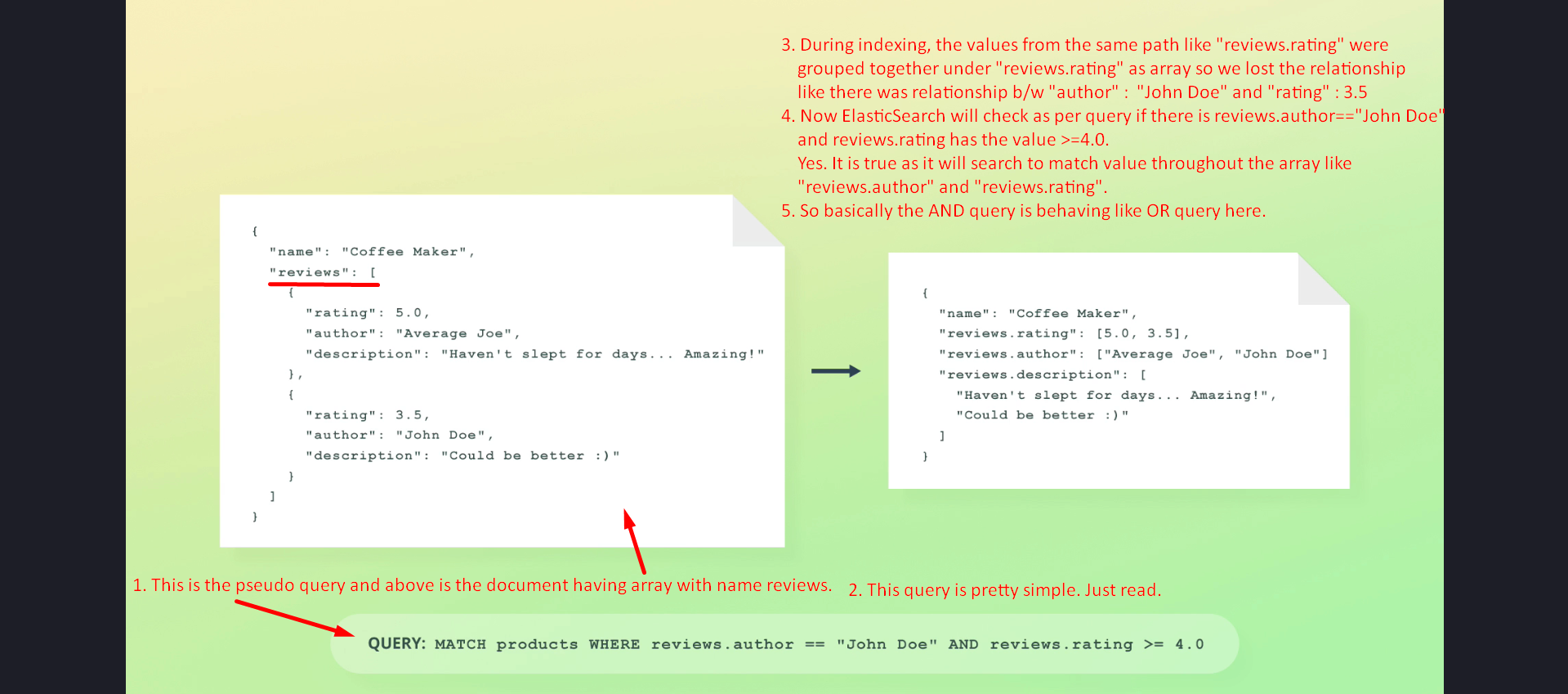
Internally, objects are flattened by Elasticsearch to ensure compatibility with Apache Lucene as Apache Lucene doesn’t support objects, using dot notation for hierarchy.  
  
Still we can query object fields using dot as if they were objects using dot notation syntax.

## 4. Challenges with Arrays of Objects

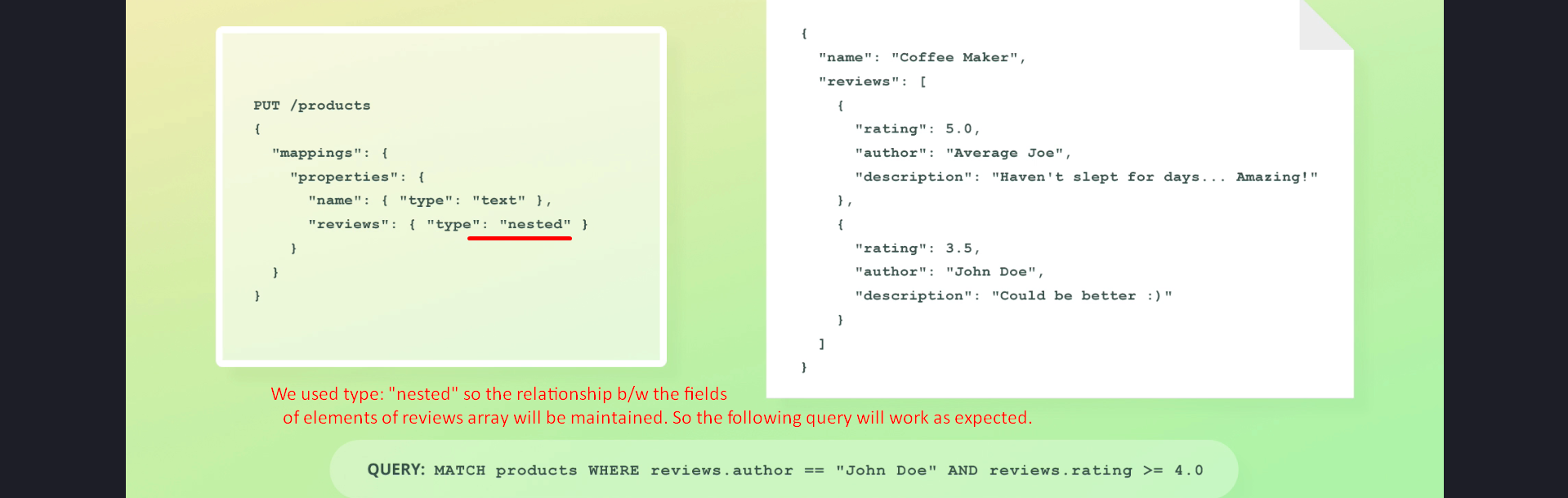
**Question**:   
What if we try to index an array of objects?  
How are the objects flattened?  
Wouldn’t the result be duplicate field names?  
**Answer**:  
The answer to the last question is Yes.

1. When indexing arrays of objects, values are grouped by field name, causing the relationships between fields to be lost.   
  
If you run a search query against one of those fields, it will search through all of the values with the array. It is nice in some cases but may create issue with some queries.

2. This can lead to unexpected query results, as the boolean AND condition may behave like a boolean OR.

3. **Example**:  
Searching for products with a review by 'John Doe' and a rating >= 4 may incorrectly match products where the relationship is not preserved.  
  
**SOLUTION**: Nested Data Type.

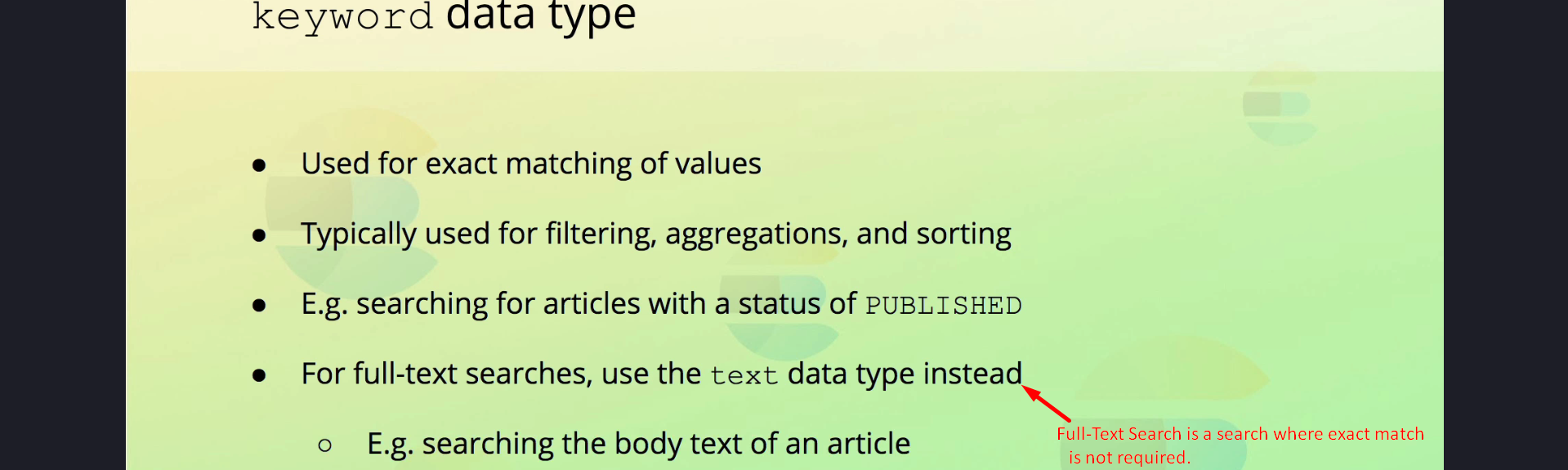
## 5. Nested Data Type

1. The `nested` data type solves the issue of lost relationships in arrays of objects by storing each object as a separate hidden document.  
  
Above 11 documents will be indexed one for product and one for each review.

2. This ensures accurate querying, maintaining relationships between object fields.

## 6. Keyword Data Type

1. The `**keyword**` **data type** is designed for fields that require exact matches, such as email addresses, order statuses, and tags.

2. This data type supports filtering, sorting, and aggregations but **not full-text searches**.  


3. Example Mapping:  
PUT /articles  
{  
 "mappings": {  
 "properties": {  
 "status": { "type": "keyword" }  
 }  
 }  
}

4. The `keyword` analyzer ensures that field values are stored unaltered, enabling exact matches.

## 7. Text Data Type

1. The `text` data type is used for full-text searches, allowing partial matches and relevance-based ranking.

2. Example: Searching for articles containing specific words within their content.

3. The distinction between `text` and `keyword` will be explored further in later sections.

## 8. Summary

1. Elasticsearch provides a variety of data types to cater to different use cases, from basic types like `integer` and `boolean` to complex ones like `object` and `nested`.

2. Specialized types like `keyword` and `text` address exact matches and full-text searches, respectively.

3. Understanding these data types and their behavior is crucial for designing efficient and accurate Elasticsearch indices.