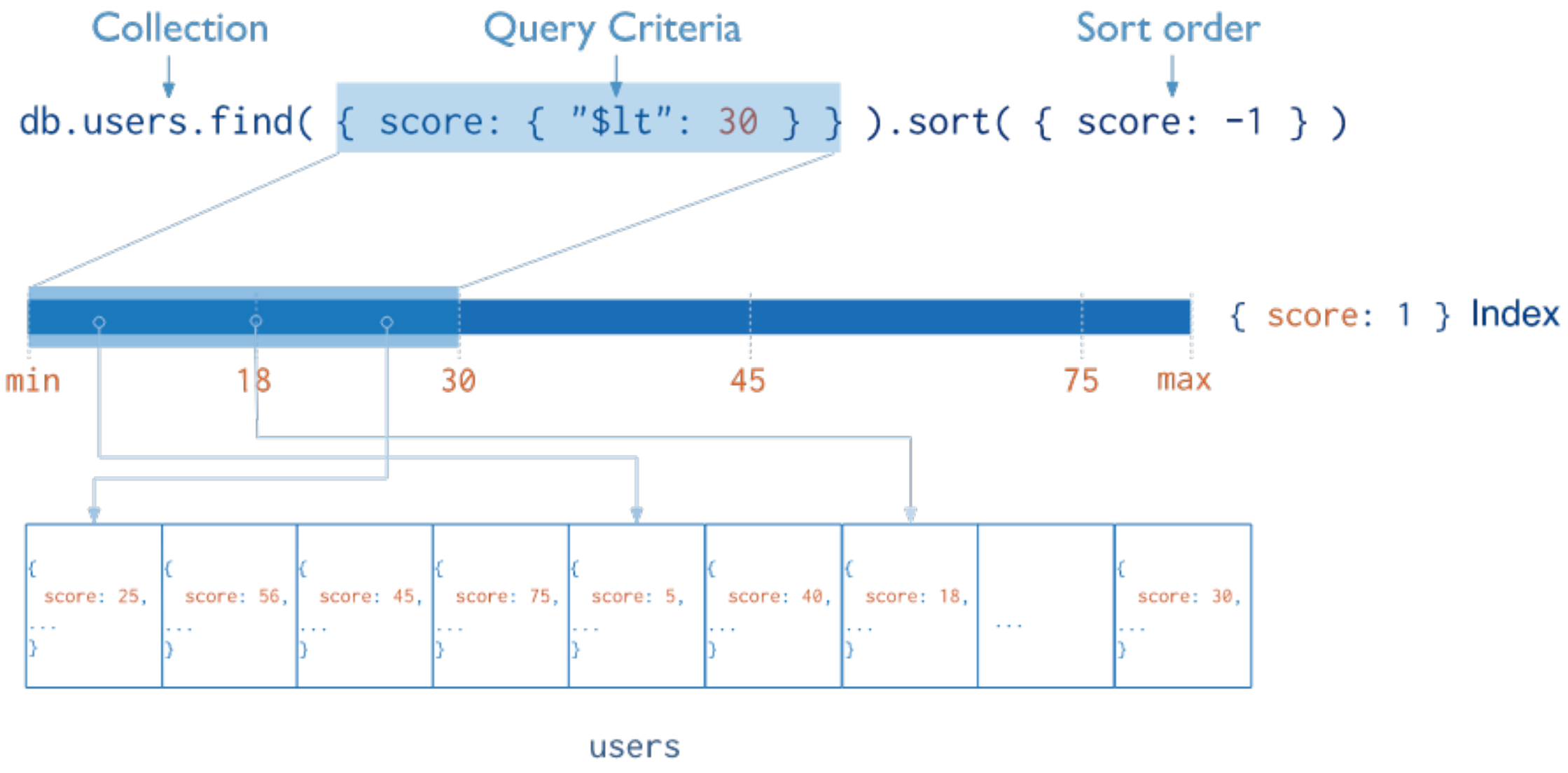
Banco de dados NoSQL - Índices

Prof. Gustavo Leitão

Índices

- Índices são utilizados para otimizar consultas
- Sem os índices o banco fará uma busca (scan) em toda coleção
- Se existir um índice apropriado à consulta o mongo o utilizará
- Comando: createIndex()

Índice de um único campo



Índice de um único campo

```
db.collection.createIndex( <key and index type specification>, <options> )
```

copy

The following example creates a single key descending index on the **name** field:

```
db.collection.createIndex( { name: -1 } )
```

copy

Índice de um único campo

Index Names

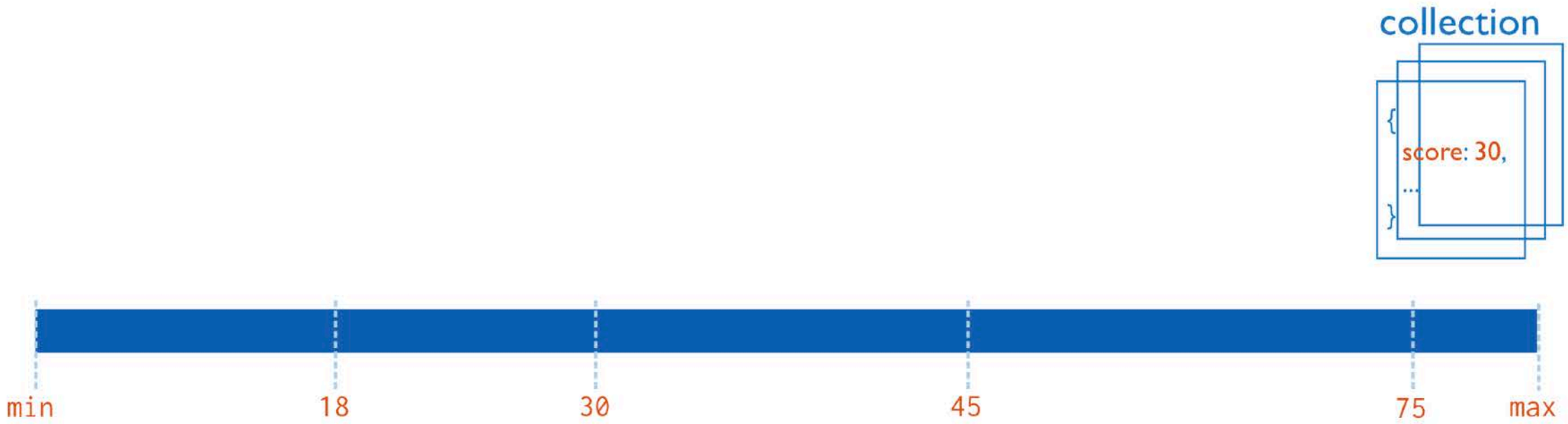
The default name for an index is the concatenation of the indexed keys and each key's direction in the index (i.e. 1 or -1) using underscores as a separator. For example, an index created on `{ item : 1, quantity: -1 }` has the name `item_1_quantity_-1`.

You can create indexes with a custom name, such as one that is more human-readable than the default. For example, consider an application that frequently queries the `products` collection to populate data on existing inventory. The following `createIndex()` method creates an index on `item` and `quantity` named `query for inventory`:

```
db.products.createIndex(
  { item: 1, quantity: -1 } ,
  { name: "query for inventory" }
)
```

Single Field

In addition to the MongoDB-defined `_id` index, MongoDB supports the creation of user-defined ascending/descending indexes on a single field of a document.

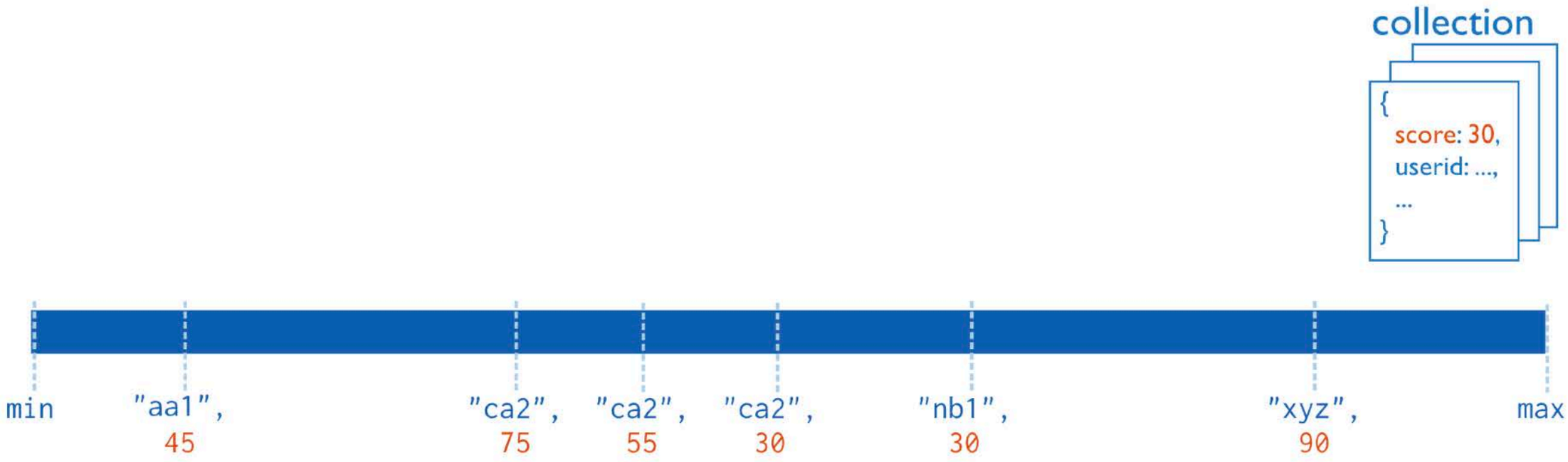


For a single-field index and sort operations, the sort order (i.e. ascending or descending) of the index key does not matter because MongoDB can traverse the index in either direction.

Compound Index

MongoDB also supports user-defined indexes on multiple fields, i.e. compound indexes.

The order of fields listed in a compound index has significance. For instance, if a compound index consists of `{ userid: 1, score: -1 }`, the index sorts first by `userid` and then, within each `userid` value, sorts by `score`.

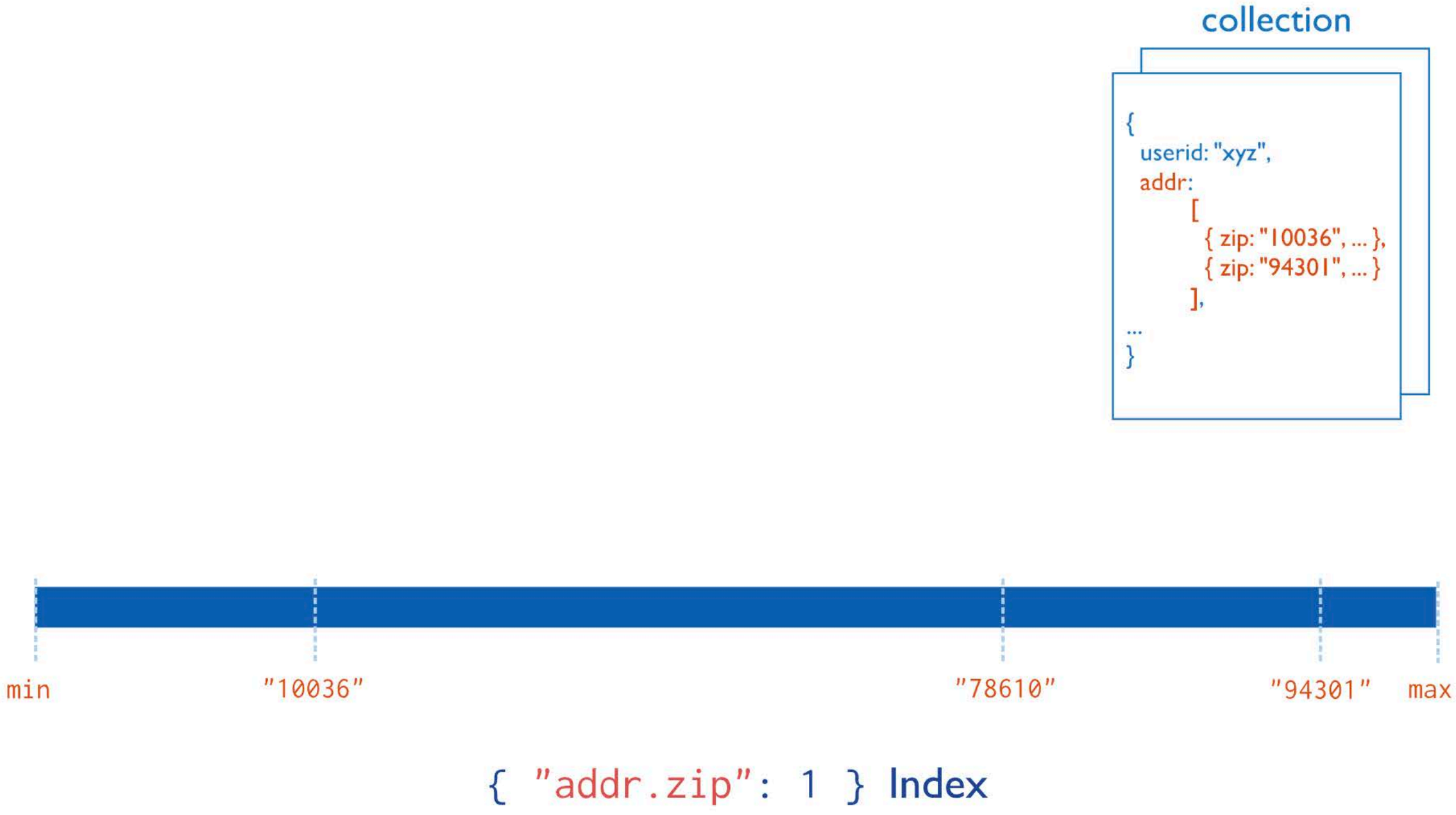


{ userid: 1, score: -1 } Index

For compound indexes and sort operations, the sort order (i.e. ascending or descending) of the index keys can determine whether the index can support a sort operation. See Sort Order for more information on the impact of index order on results in compound indexes.

Multikey Index

MongoDB uses multikey indexes to index the content stored in arrays. If you index a field that holds an array value, MongoDB creates separate index entries for *every* element of the array. These multikey indexes allow queries to select documents that contain arrays by matching on element or elements of the arrays. MongoDB automatically determines whether to create a multikey index if the indexed field contains an array value; you do not need to explicitly specify the multikey type.



Geospatial Index

To support efficient queries of geospatial coordinate data, MongoDB provides two special indexes: 2d indexes that uses planar geometry when returning results and 2dsphere indexes that use spherical geometry to return results.

See 2d Index Internals for a high level introduction to geospatial indexes.

Text Indexes

MongoDB provides a `text` index type that supports searching for string content in a collection. These text indexes do not store language-specific *stop* words (e.g. “the”, “a”, “or”) and *stem* the words in a collection to only store root words.

See Text Indexes for more information on text indexes and search.

The following examples assume a collection `articles` that has a version 3 text index on the field `subject`:

copy

```
db.articles.createIndex( { subject: "text" } )
```

Populate the collection with the following documents:

copy

```
db.articles.insert(
   [
     { _id: 1, subject: "coffee", author: "xyz", views: 50 },
     { _id: 2, subject: "Coffee Shopping", author: "efg", views: 5 },
     { _id: 3, subject: "Baking a cake", author: "abc", views: 90  },
     { _id: 4, subject: "baking", author: "xyz", views: 100 },
     { _id: 5, subject: "Café Con Leche", author: "abc", views: 200 },
     { _id: 6, subject: "Сырники", author: "jkl", views: 80 },
     { _id: 7, subject: "coffee and cream", author: "efg", views: 10 },
     { _id: 8, subject: "Cafe con Leche", author: "xyz", views: 10 }
   ]
)
```

Search for a Single Word

The following query specifies a `$search` string of `coffee`:

```
db.articles.find( { $text: { $search: "coffee" } } )
```

copy

This query returns the documents that contain the term `coffee` in the indexed `subject` field, or more precisely, the stemmed version of the word:

```
{ "_id" : 2, "subject" : "Coffee Shopping", "author" : "efg", "views" : 5 }
{ "_id" : 7, "subject" : "coffee and cream", "author" : "efg", "views" : 10 }
{ "_id" : 1, "subject" : "coffee", "author" : "xyz", "views" : 50 }
```

copy

Match Any of the Search Terms

If the search string is a space-delimited string, `$text` operator performs a logical `OR` search on each term and returns documents that contains any of the terms.

The following query specifies a `$search` string of three terms delimited by space, `"bake coffee cake"`:

```
db.articles.find( { $text: { $search: "bake coffee cake" } } )
```

This query returns documents that contain either `bake` or `coffee` or `cake` in the indexed `subject` field, or more precisely, the stemmed version of these words:

```
{ "_id" : 2, "subject" : "Coffee Shopping", "author" : "efg", "views" : 5 }
{ "_id" : 7, "subject" : "coffee and cream", "author" : "efg", "views" : 10 }
{ "_id" : 1, "subject" : "coffee", "author" : "xyz", "views" : 50 }
{ "_id" : 3, "subject" : "Baking a cake", "author" : "abc", "views" : 90 }
{ "_id" : 4, "subject" : "baking", "author" : "xyz", "views" : 100 }
```

Search for a Phrase

To match the exact phrase as a single term, escape the quotes.

The following query searches for the phrase `coffee shop`:

```
db.articles.find( { $text: { $search: "\"coffee shop\"" } } )
```

copy

This query returns documents that contain the phrase `coffee shop`:

```
{ "_id" : 2, "subject" : "Coffee Shopping", "author" : "efg", "views" : 5 }
```

copy

Exclude Documents That Contain a Term

A *negated* term is a term that is prefixed by a minus sign `-`. If you negate a term, the `$text` operator will exclude the documents that contain those terms from the results.

The following example searches for documents that contain the words `coffee` but do **not** contain the term `shop`, or more precisely the stemmed version of the words:

```
db.articles.find( { $text: { $search: "coffee -shop" } } )
```

The query returns the following documents:

```
{ "_id" : 7, "subject" : "coffee and cream", "author" : "efg", "views" : 10 }
{ "_id" : 1, "subject" : "coffee", "author" : "xyz", "views" : 50 }
```

Search a Different Language

Use the optional `$language` field in the `$text` expression to specify a language that determines the list of stop words and the rules for the stemmer and tokenizer for the search string.

If you specify a language value of `"none"`, then the text search uses simple tokenization with no list of stop words and no stemming.

The following query specifies `es`, i.e. Spanish, as the language that determines the tokenization, stemming, and stop words:

```
db.articles.find(
   { $text: { $search: "leche", $language: "es" } }
)
```

The query returns the following documents:

```
{ "_id" : 5, "subject" : "Café Con Leche", "author" : "abc", "views" : 200 }
{ "_id" : 8, "subject" : "Cafe con Leche", "author" : "xyz", "views" : 10 }
```

Case and Diacritic Insensitive Search

Changed in version 3.2.

The `$text` operator defers to the case and diacritic insensitivity of the `text` index. The version 3 `text` index is diacritic insensitive and expands its case insensitivity to include the Cyrillic alphabet as well as characters with diacritics. For details, see text Index Case Insensitivity and text Index Diacritic Insensitivity.

The following query performs a case and diacritic insensitive text search for the terms сы́рники or CAFÉS:

```
db.articles.find( { $text: { $search: "сы́рники CAFÉS" } } )
```

Using the version 3 `text` index, the query matches the following documents.

```
{ "_id" : 6, "subject" : "Сырники", "author" : "jkl", "views" : 80 }
{ "_id" : 5, "subject" : "Café Con Leche", "author" : "abc", "views" : 200 }
{ "_id" : 8, "subject" : "Cafe con Leche", "author" : "xyz", "views" : 10 }
```

Index Properties

Unique Indexes

The unique property for an index causes MongoDB to reject duplicate values for the indexed field. Other than the unique constraint, unique indexes are functionally interchangeable with other MongoDB indexes.

Partial Indexes

New in version 3.2.

Partial indexes only index the documents in a collection that meet a specified filter expression. By indexing a subset of the documents in a collection, partial indexes have lower storage requirements and reduced performance costs for index creation and maintenance.

Partial indexes offer a superset of the functionality of sparse indexes and should be preferred over sparse indexes.

Sparse Indexes

The sparse property of an index ensures that the index only contain entries for documents that have the indexed field. The index skips documents that *do not* have the indexed field.

You can combine the sparse index option with the unique index option to prevent inserting documents that have duplicate values for the indexed field(s) and skip indexing documents that lack the indexed field(s).

TTL Indexes

On this page

TTL indexes are special single-field indexes that MongoDB can use to automatically remove documents from a collection after a certain amount of time or at a specific clock time. Data expiration is useful for certain types of information like machine generated event data, logs, and session information that only need to persist in a database for a finite amount of time.

To create a TTL index, use the `db.collection.createIndex()` method with the `expireAfterSeconds` option on a field whose value is either a date or an array that contains date values.

For example, to create a TTL index on the `lastModifiedDate` field of the `eventlog` collection, use the following operation in the `mongo` shell:

```
db.eventlog.createIndex( { "lastModifiedDate": 1 }, { expireAfterSeconds: 3600 } )
```

Índices

- Por padrão, o Mongo cria **índice único** pelo campo _id.
- Com isso, o banco garante que não possa existir dois documentos com mesmo _id

Exercício

- Desenvolva um pequeno programa que crie documentos e salve em uma coleção e siga os seguintes pontos:
    - O documento deve possuir dois campos (val1 e val2) numéricos com valores aleatórios de 0 a 100.
    - Gere e insira pelo menos 1 milhão de documentos (meça o tempo de inserção total)
    - Realize uma consulta por valores em val1 entre 0 e 10 e meça o tempo.
    - Crie um indice pelo campo val1
    - Repita a consulta anterior medindo o tempo. O que ocorre?
    - Agora repita a consulta anterior retornando apenas o campo val1 (utilize projeção para remover o _id e val2). O que ocorre?
    - Insira mais 1 milhão de registros e meça o tempo de inserção comparando com o valor obtido antes do índice.

Banco de dados NoSQL - Índices

Prof. Gustavo Leitão