The following diagram illustrates a query that selects and orders the matching documents using an index:

Fundamentally, indexes in MongoDB are similar to indexes in other database systems. MongoDB defines indexes at the [collection](https://docs.mongodb.com/manual/reference/glossary/#term-collection) level and supports indexes on any field or sub-field of the documents in a MongoDB collection.

Default \_id Index

MongoDB creates a [unique index](https://docs.mongodb.com/manual/core/index-unique/#index-type-unique) on the [\_id](https://docs.mongodb.com/manual/core/document/#document-id-field) field during the creation of a collection. The \_id index prevents clients from inserting two documents with the same value for the \_id field. You cannot drop this index on the \_id field.

**NOTE**

In [sharded clusters](https://docs.mongodb.com/manual/reference/glossary/#term-sharded-cluster), if you do *not* use the \_id field as the [shard key](https://docs.mongodb.com/manual/reference/glossary/#term-shard-key), then your application **must** ensure the uniqueness of the values in the \_id field to prevent errors. This is most-often done by using a standard auto-generated [ObjectId](https://docs.mongodb.com/manual/reference/glossary/#term-objectid).

Create an Index

To create an index, use [db.collection.createIndex()](https://docs.mongodb.com/manual/reference/method/db.collection.createIndex/#db.collection.createIndex) or a similar [method from your driver](https://api.mongodb.com/).

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

The [db.collection.createIndex()](https://docs.mongodb.com/manual/reference/method/db.collection.createIndex/#db.collection.createIndex) method only creates an index if an index of the same specification does not already exist.

|  |  |
| --- | --- |
| [[1]](https://docs.mongodb.com/manual/indexes/index.html#id2) | MongoDB indexes use a B-tree data structure. |

Index Types

MongoDB provides a number of different index types to support specific types of data and queries.

**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](https://docs.mongodb.com/manual/core/index-single/).

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.

See [Single Field Indexes](https://docs.mongodb.com/manual/core/index-single/) and [Sort with a Single Field Index](https://docs.mongodb.com/manual/tutorial/sort-results-with-indexes/#sort-results-single-field) for more information on single-field indexes.

**Compound Index**

MongoDB also supports user-defined indexes on multiple fields, i.e. [compound indexes](https://docs.mongodb.com/manual/core/index-compound/).

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.

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](https://docs.mongodb.com/manual/core/index-compound/#index-ascending-and-descending) for more information on the impact of index order on results in compound indexes.

See [Compound Indexes](https://docs.mongodb.com/manual/core/index-compound/) and [Sort on Multiple Fields](https://docs.mongodb.com/manual/tutorial/sort-results-with-indexes/#sort-on-multiple-fields) for more information on compound indexes.

**Multikey Index**

MongoDB uses [multikey indexes](https://docs.mongodb.com/manual/core/index-multikey/) 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](https://docs.mongodb.com/manual/core/index-multikey/) 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.

See [Multikey Indexes](https://docs.mongodb.com/manual/core/index-multikey/) and [Multikey Index Bounds](https://docs.mongodb.com/manual/core/multikey-index-bounds/) for more information on multikey indexes.

**Geospatial Index**

To support efficient queries of geospatial coordinate data, MongoDB provides two special indexes: [2d indexes](https://docs.mongodb.com/manual/core/2d/)that uses planar geometry when returning results and [2dsphere indexes](https://docs.mongodb.com/manual/core/2dsphere/) that use spherical geometry to return results.

See [2d Index Internals](https://docs.mongodb.com/manual/core/geospatial-indexes/) 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](https://docs.mongodb.com/manual/core/index-text/) for more information on text indexes and search.

**Hashed Indexes**

To support [hash based sharding](https://docs.mongodb.com/manual/core/hashed-sharding/#sharding-hashed-sharding), MongoDB provides a [hashed index](https://docs.mongodb.com/manual/core/index-hashed/) type, which indexes the hash of the value of a field. These indexes have a more random distribution of values along their range, but *only* support equality matches and cannot support range-based queries.

Index Properties

**Unique Indexes**

The [unique](https://docs.mongodb.com/manual/core/index-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](https://docs.mongodb.com/manual/core/index-partial/) 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](https://docs.mongodb.com/manual/core/index-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 reject documents that have duplicate values for a field but ignore documents that do not have the indexed key.

**TTL Indexes**

[TTL indexes](https://docs.mongodb.com/manual/core/index-ttl/) are special indexes that MongoDB can use to automatically remove documents from a collection after a certain amount of time. This is ideal 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.

See: [Expire Data from Collections by Setting TTL](https://docs.mongodb.com/manual/tutorial/expire-data/) for implementation instructions.

Index Use

Indexes can improve the efficiency of read operations. The [Analyze Query Performance](https://docs.mongodb.com/manual/tutorial/analyze-query-plan/) tutorial provides an example of the execution statistics of a query with and without an index.

For information on how MongoDB chooses an index to use, see [query optimizer](https://docs.mongodb.com/manual/core/query-plans/#read-operations-query-optimization).

Indexes and Collation

To use an index for string comparisons, an operation must also specify the same collation. That is, an index with a collation cannot support an operation that performs string comparisons on the indexed fields if the operation specifies a different collation.

For example, the collection myColl has an index on a string field category with the collation locale "fr".

db.myColl.createIndex( { category: 1 }, { collation: { locale: "fr" } } )

The following query operation, which specifies the same collation as the index, can use the index:

db.myColl.find( { category: "cafe" } ).collation( { locale: "fr" } )

However, the following query operation, which by default uses the “simple” binary collator, cannot use the index:

db.myColl.find( { category: "cafe" } )

For a compound index where the index prefix keys are not strings, arrays, and embedded documents, an operation that specifies a different collation can still use the index to support comparisons on the index prefix keys.

For example, the collection myColl has a compound index on the numeric fields score and price and the string field category; the index is created with the collation locale "fr" for string comparisons:

db.myColl.createIndex(

{ score: 1, price: 1, category: 1 },

{ collation: { locale: "fr" } } )

The following operations, which use "simple" binary collation for string comparisons, can use the index:

db.myColl.find( { score: 5 } ).sort( { price: 1 } )

db.myColl.find( { score: 5, price: { $gt: NumberDecimal( "10" ) } } ).sort( { price: 1 } )

The following operation, which uses "simple" binary collation for string comparisons on the indexed category field, can use the index to fulfill only the score: 5 portion of the query:

db.myColl.find( { score: 5, category: "cafe" } )

Covered Queries

When the query criteria and the [projection](https://docs.mongodb.com/manual/reference/glossary/#term-projection) of a query include *only* the indexed fields, MongoDB will return results directly from the index *without* scanning any documents or bringing documents into memory. These covered queries can be *very* efficient.

For more information on covered queries, see [Covered Query](https://docs.mongodb.com/manual/core/query-optimization/#read-operations-covered-query).

Index Intersection

*New in version 2.6.*

MongoDB can use the [intersection of indexes](https://docs.mongodb.com/manual/core/index-intersection/) to fulfill queries. For queries that specify compound query conditions, if one index can fulfill a part of a query condition, and another index can fulfill another part of the query condition, then MongoDB can use the intersection of the two indexes to fulfill the query. Whether the use of a compound index or the use of an index intersection is more efficient depends on the particular query and the system.

For details on index intersection, see [Index Intersection](https://docs.mongodb.com/manual/core/index-intersection/).

Restrictions

Certain restrictions apply to indexes, such as the length of the index keys or the number of indexes per collection. See [Index Limitations](https://docs.mongodb.com/manual/reference/limits/#index-limitations) for details.