CHECKLIST

MongoDB Indexing Checklist for Query Performance Optimization

A practical checklist for planning, creating, validating, and operating MongoDB indexes before production use.

MongoDB Indexing Checklist for Query Performance Optimization

Use this checklist to plan, implement, validate, and operate a MongoDB index for a specific query pattern.

---

1) Confirm the problem is an indexing problem

- [ ] I have a specific slow query, aggregation stage, or application code path to optimize.
- [ ] I have evidence from logs, APM, profiling, or explain() output.
- [ ] I have ruled out non-index bottlenecks such as:
- [ ] large result sets returned to the application
- [ ] expensive client-side processing
- [ ] network latency
- [ ] missing projection causing excessive field retrieval
- [ ] heavy post-filter computation in an aggregation pipeline
- [ ] I can describe the query?s main filter, sort, and join behavior.

---

2) Document the target query shape

- [ ] I have the exact filter fields used by the query.
- [ ] I have identified which predicates are equality matches.
- [ ] I have identified which predicates are range conditions.
- [ ] I know whether the query sorts results, and if so, by which field(s).
- [ ] I know whether the query must support exact match, range scan, sort support, or a combination.
- [ ] I know whether the query returns a narrow result set or a broad slice of data.
- [ ] I can express the query in representative form, for example:

---

3) Check prerequisites before building an index

- [ ] I know the expected cardinality of the indexed fields.
- [ ] I understand the selectivity of the query predicates.
- [ ] I know the acceptable write overhead for the collection.
- [ ] I have a production-like staging environment, or I understand the risk of testing directly in production.
- [ ] I have permission to inspect query plans and create indexes.

---

4) Design the index

- [ ] Equality fields are placed first in the compound index.
- [ ] Range fields are placed after equality fields.
- [ ] Sort fields are aligned with the index order when the index should satisfy sorting.

- [ ] The index is as narrow as possible while still supporting the query.
- [ ] The index does not begin with a low-selectivity field unless that field is always part of the query.
- [ ] The index definition matches the actual query pattern rather than a hypothetical one.

Example candidate:

---

5) Check for existing or overlapping indexes

- [ ] I reviewed the collection?s current indexes.
- [ ] I checked for identical prefixes that may already support the query.
- [ ] I checked whether an existing index already supports the sort.
- [ ] I identified any partially overlapping indexes.
- [ ] I determined whether the candidate is:
- [ ] new and needed
- [ ] redundant
- [ ] replaceable by an existing index
- [ ] too expensive for the expected gain
- [ ] I considered index consolidation to avoid index proliferation.

---

6) Validate the candidate with explain()

- [ ] I ran the target query through explain() before making changes.
- [ ] I noted whether the current plan uses a collection scan.
- [ ] I noted whether the current plan uses a blocking sort.
- [ ] I noted documents examined vs. documents returned.
- [ ] I created a candidate index only after comparing it against current behavior.

- [ ] I have a before-and-after baseline for comparison.

What to look for after indexing:

- [ ] index scan instead of collection scan
- [ ] fewer documents examined
- [ ] no blocking sort when the index should support ordering
- [ ] stable plan selection with representative inputs

---

7) Create the index safely

- [ ] I selected a controlled change window when possible.
- [ ] I understand the index build behavior for my MongoDB version and topology.
- [ ] I assessed the impact of index build time on CPU, I/O, and memory.
- [ ] I confirmed the index build strategy is appropriate for a hot or large collection.
- [ ] I created the index with the intended definition.

Example:

- [ ] I verified the index exists after creation.
- [ ] I confirmed there were no unexpected application errors or write failures.
- [ ] I confirmed there was no unacceptable latency spike during the build.

---

8) Validate query performance after deployment

- [ ] The query now uses the intended index or another clearly appropriate index.
- [ ] The winning plan is better than the baseline.
- [ ] Query latency improved for representative inputs.
- [ ] The improvement holds for different tenants, date ranges, or other realistic parameter sets.

- [ ] The plan remains stable under normal workload variation.
- [ ] The sort is satisfied by the index when that was part of the goal.

---

9) Check for operational side effects

- [ ] I measured write overhead after adding the index.
- [ ] I reviewed index size and growth rate.
- [ ] I confirmed the index does not create unacceptable storage pressure.
- [ ] I checked whether the index increases maintenance cost during inserts, updates, or deletes.
- [ ] I verified the index is still useful for production traffic, not just synthetic tests.

---

10) Review for index drift over time

- [ ] I have a process for periodically reviewing query plans.
- [ ] I can identify when a query shape has changed.
- [ ] I can detect when an index is no longer useful.
- [ ] I have a plan to retire redundant indexes safely.
- [ ] I revisit index design when application filters, sorts, or aggregations change.

---

11) Common failure checks

Stop and re-evaluate if any of these are true:

- [ ] The query target is not clearly defined.
- [ ] The main bottleneck is not actually indexing-related.

- [ ] The compound field order does not match the query shape.
- [ ] The index exists but the query still performs a collection scan.
- [ ] The query still requires a blocking sort.
- [ ] The new index duplicates an existing one.
- [ ] The write cost or storage cost is too high for the benefit gained.

---

12) Production readiness checklist

Before calling the index ready for production, confirm:

- [ ] The query is documented.
- [ ] The index design is justified.
- [ ] Existing indexes were reviewed.
- [ ] The index was built safely.
- [ ] explain() confirms the desired plan.
- [ ] Realistic workloads were tested.
- [ ] Write overhead was reviewed.
- [ ] Index size and growth were reviewed.
- [ ] Ongoing monitoring and cleanup are defined.

Quick decision summary

Use a MongoDB index when:

- the query is clearly identified
- the index matches filter and sort behavior
- explain plans show reduced scanning
- operational impact is acceptable

Do not add an index when:

- the query pattern is unknown
- another bottleneck is the real issue
- the candidate is redundant or poorly ordered
- the write and storage costs outweigh the benefit

Final sign-off

- [ ] I confirmed the index improves the target query.
- [ ] I confirmed the operational impact is acceptable.
- [ ] I documented the final index definition and rationale.
- [ ] I have a follow-up plan to monitor performance after deployment.