Indexing Use Case for CSC

---

0. CSC Use Case – What Data Are We Indexing?

Imagine CSC is managing **thousands of client entities** across states:

- **entities** – each legal entity CSC manages
- **filings** – annual reports, franchise tax filings, etc.
- **reminders** – reminders sent to clients before due dates

We'll use a database:

```
use csc_compliance
```

Collections & Sample Documents

entities

```
db.entities.insertMany([
  {
    _id: 1001,
    name: "Acme Holdings Inc.",
    clientId: "CLI-001",
    jurisdiction: "DE",
    status: "Active",
    industry: "FinTech",
    serviceTypes: ["registered_agent", "annual_report"], //
multikey
    tax_id: "99-1234567", // optional, not all entities have
it
    notes: "Key client in Delaware"
  },
  {
    _id: 1002,
    name: "Sunrise Logistics LLC",
    clientId: "CLI-002",
    jurisdiction: "CA",
    status: "Active",
    industry: "Logistics",
    serviceTypes: ["registered_agent"],
    notes: "West coast expansion client"
  }
])
```

filings

```
db.filings.insertMany([
  {
    _id: 1,
    entity_id: 1001,
    filing_type: "Annual Report",
    state: "DE",
    due_date: ISODate("2025-03-01"),
    filed_date: null,
    status: "OPEN",
    amount: 5000
  },
  {
    _id: 2,
    entity_id: 1001,
    filing_type: "Franchise Tax",
    state: "DE",
    due_date: ISODate("2025-03-15"),
    filed_date: null,
    status: "OPEN",
    amount: 20000
  },
  {
    _id: 3,
    entity_id: 1002,
    filing_type: "Annual Report",
    state: "CA",
    due_date: ISODate("2025-02-10"),
    filed_date: ISODate("2025-02-01"),
    status: "FILED",
    amount: 1500
  }
])
```

reminders

```
db.reminders.insertOne({
  entity_id: 1001,
  filing_id: 1,
  channel: "email",
  status: "SENT",
  createdAt: new Date(),
  expiresAt: new Date(Date.now() + 30*24*60*60*1000) // 30
days later
```

```
})
```

---

1. MongoDB Indexes – Basics

1.1 Why Indexes Matter – CSC Use Case

Business Question:

"Show all **OPEN Delaware filings due in the next 30 days** so our compliance team can call those clients."

Step 1: Query without index

```
db.filings.explain("executionStats").find({
  state: "DE",
  status: "OPEN",
  due_date: { $lte: ISODate("2025-03-31") }
})
```

Look at executionStats:

- executionStages.stage → usually "COLLSCAN"
- totalDocsExamined → may be large in real data

Explanation: MongoDB is scanning *every* document in filings.

On a real CSC dataset with millions of filings, this is **slow**.

1.2 Single-Field Index – Hands-On

Use Case

Most queries filter by **state** or **entity_id**:

- "List all filings in Delaware."
- "Show all filings for entity 1001."

Step 2: Create a single-field index on state

```
db.filings.createIndex({ state: 1 })
```

Step 3: Re-run the query with explain

```
db.filings.explain("executionStats").find({
  state: "DE",
```

```
  status: "OPEN",
  due_date: { $lte: ISODate("2025-03-31") }
})
```

Now you should see:

- stage → "IXSCAN" + "FETCH"
- totalKeysExamined <<< total docs in collection

👉 **Impact:** Query is now using the index to jump directly to DE filings.

1.3 Query Planner Introduction

Key things to show participants in explain():

- queryPlanner.winningPlan.stage – COLLSCAN vs IXSCAN
- indexName – which index is used
- executionStats.totalDocsExamined vs nReturned

Tip: *Compare before & after index* – that visual impact teaches "why indexes matter".

1.4 Lab 1 – Create & Test Index Performance (Step-by-step)

1. **Seed data**
   - Insert ~10k dummy filings using a loop (or scripted JSON).
2. **Run query without index**
   - Use the "DE + OPEN + due_date" filter.
   - Capture totalDocsExamined, executionTimeMillis.
3. **Create index**

```
db.filings.createIndex({ state: 1 })
```

3.
4. **Re-run explain()**
   - Compare COLLSCAN → IXSCAN.
5. **Optional:** Create index on entity_id

```
db.filings.createIndex({ entity_id: 1 })
```

5. Query filings for one entity to see speed improvement.

---

2. MongoDB Indexes in Detail – CSC-Focused

Now we go deeper: compound, multikey, text, TTL, partial, sparse, monitoring.

2.1 Compound Indexes – state + due_date

Use Case

"Show **OPEN filings in Delaware, ordered by due date**, for the next 30 days."

Query:

```
db.filings.find({
  state: "DE",
  status: "OPEN",
  due_date: { $lte: ISODate("2025-03-31") }
}).sort({ due_date: 1 })
```

Step 1: Create compound index

```
db.filings.createIndex({ state: 1, due_date: 1 })
```

Step 2: Explain

```
db.filings.explain("executionStats").find({
  state: "DE",
  status: "OPEN",
  due_date: { $lte: ISODate("2025-03-31") }
}).sort({ due_date: 1 })
```

You should see:

- IXSCAN on { state: 1, due_date: 1 }
- No extra in-memory sort ("stage": "SORT" should disappear)

Explanation to participants:

- Equality field (state) should come before range (due_date) – classic rule.
- Compound index also supports prefix queries like { state: "DE" }.

2.2 Multikey Indexes – serviceTypes Array

Use Case

CSC wants to find all **entities subscribed to "annual_report" service**:

```
db.entities.find({ serviceTypes: "annual_report" })
```

Step 1: Create multikey index

```
db.entities.createIndex({ serviceTypes: 1 })
```

MongoDB automatically creates a **multikey index** because serviceTypes is an array.

Step 2: Test & Explain

```
db.entities.explain("executionStats").find({ serviceTypes:
"annual_report" })
```

Explanation:

- Each array element becomes an index entry.
- Great for tags, services, labels (typical in CSC client profiles).

2.3 Text Indexes – Search by Entity Name & Notes

Use Case

CSC account manager searches:

"Find all entities with 'Holdings' or 'Delaware' in name/notes."

Step 1: Create text index

```
db.entities.createIndex({
  name: "text",
  notes: "text"
})
```

Step 2: Run full-text search

```
db.entities.find({
  $text: { $search: "Holdings Delaware" }
```

```
}, {
  score: { $meta: "textScore" },
  name: 1,
  notes: 1
}).sort({ score: { $meta: "textScore" } })
```

Explanation:

- Text index tokenizes strings → efficient search.
- Perfect for CSC's **client name search**, **notes search** in dashboards.

2.4 TTL Indexes – Auto-Cleaning Old Reminders

Use Case

CSC only wants **reminders** to exist for 30 days after they are created or expire.

We already inserted:

```
{
  entity_id: 1001,
  filing_id: 1,
  channel: "email",
  status: "SENT",
  createdAt: new Date(),
  expiresAt: <Date+30d>
}
```

Option A – TTL using

createdAt

```
db.reminders.createIndex(
  { createdAt: 1 },
  { expireAfterSeconds: 30*24*60*60 } // 30 days
)
```

Option B – TTL using expiresAt

(explicit expiry date)

```
db.reminders.createIndex(
  { expiresAt: 1 },
  { expireAfterSeconds: 0 } // expireAt is the exact deletion
time
)
```

Explanation:

- MongoDB's TTL monitor will delete docs **automatically** after threshold.
- Great for **temporary tokens, reminders, session data**.

2.5 Partial Indexes – Only "OPEN" Filings

Use Case

Most queries in CSC UI focus on **OPEN filings**, not older FILED/CLOSED ones.

Typical query:

```
db.filings.find({
  status: "OPEN",
  state: "DE",
  due_date: { $lte: ISODate("2025-03-31") }
})
```

Step 1: Create partial index

```
db.filings.createIndex(
  { state: 1, due_date: 1 },
  { partialFilterExpression: { status: "OPEN" } }
)
```

What happens:

- Index contains **only documents where status: "OPEN"**.
- Smaller index size → faster, cheaper, better cache hit.

Step 2: Explain

```
db.filings.explain("executionStats").find({
  status: "OPEN",
  state: "DE",
  due_date: { $lte: ISODate("2025-03-31") }
})
```

You should see the partial index in queryPlanner.winningPlan.

Explain in class:

"We don't waste index space on closed history; we optimize for what business *actually* queries."

2.6 Sparse Indexes – Optional tax_id

Use Case

Only some CSC entities have a **tax_id**. Others (for foreign entities, old data) may not.

Searching for an entity by tax_id:

```
db.entities.find({ tax_id: "99-1234567" })
```

Step 1: Create sparse index

```
db.entities.createIndex(
  { tax_id: 1 },
  { sparse: true }
)
```

Behavior:

- Only docs *with* tax_id are included in the index.
- Docs without tax_id are ignored → smaller index.

Warning for training:

- Be careful with unique + sparse.
- Multiple docs *without* the field are allowed (not treated as duplicates).

2.7 Index Performance Monitoring – Finding Slow Queries

Basic Tools (Self-Managed or Local)

1. **Collection stats**

```
db.filings.stats()
db.filings.getIndexes()
```

1.
2. **Profiler** – Find slow queries

```
// enable profiling for slow ops > 50ms
db.setProfilingLevel(1, { slowms: 50 })
```

2. Then run your app/queries, and check:

```
db.system.profile.find().sort({ ts: -1 }).limit(5).pretty()
```

2.
3. **Explain plans** – we already used:

```
db.filings.explain("executionStats").find({ ... })
```

Story to tell:

CSC's DBA runs profiler, finds **frequent COLLSCANs** on filings; they design new indexes to match those filters.

2.8 Lab 2 – Identify Slow Queries & Fix Them

Goal: Make trainees think like CSC performance engineers.

Step 1: Enable Profiler

```
db.setProfilingLevel(1, { slowms: 20 }) // low for lab
```

Step 2: Run some "bad" queries (no proper indexes)

Examples:

```
// 1) Filter on status + amount
db.filings.find({
  status: "OPEN",
  amount: { $gt: 10000 }
})

// 2) Search entities by industry + jurisdiction
db.entities.find({
```

```
  industry: "FinTech",
  jurisdiction: "DE"
})
```

Step 3: Inspect slow queries

```
db.system.profile.find().sort({ ts: -1 }).limit(10).pretty()
```

Have participants:

- Identify which collection was slow.
- Note the filter fields in the query object.

Step 4: Design appropriate indexes

For the slow queries above:

```
// On filings
db.filings.createIndex({ status: 1, amount: 1 })

// On entities
db.entities.createIndex({ industry: 1, jurisdiction: 1 })
```

Step 5: Re-run queries with

explain("executionStats")

- Confirm IXSCAN instead of COLLSCAN.
- See reduction in totalDocsExamined and executionTimeMillis.

Step 6: Discussion

Have trainees answer:

- Why did we choose these fields in the index?
- Could any of these benefit from **partial** or **compound** design?
- Are there low-cardinality fields that *shouldn't* be indexed (e.g., status alone)?

Recap – Index Strategy for CSC

1. **Single-field:** quick wins on state, entity_id, clientId.
2. **Compound:** match common filters + sort (e.g., { state, due_date }).
3. **Multikey:** arrays like serviceTypes.
4. **Text:** searching by name / notes.
5. **TTL:** auto-cleaning reminders, tokens.
6. **Partial:** focus on status: "OPEN" or high-value filings.
7. **Sparse:** optional fields like tax_id.
8. **Monitoring:** profiler + explain + stats to guide index design.
