

Here is the practical guide, formatted for easy copying into a Microsoft Word document.

1. Start & Switch DB

```
mongosh
```

```
use shopDB
```

2. Create Sample Data

products (master product list)

```
db.products.drop();
```

```
db.products.insertMany([
```

```
  { _id: 101, name: "USB Cable", category: "Accessories", price: 199 },
  { _id: 102, name: "Wireless Mouse", category: "Accessories", price: 899 },
  { _id: 103, name: "Mechanical Keyboard", category: "Accessories", price:
3200 },
  { _id: 201, name: "Smartphone X", category: "Mobile", price: 45000 },
  { _id: 202, name: "Tablet Pro", category: "Tablet", price: 35000 }
```

```
]);
```

orders (each order contains an array of items)

```
db.orders.drop();
```

```
db.orders.insertMany([
```

```
{
```

```
order_id: "O1001",
customer: { id: 1, name: "Amit" },
date: new Date("2025-10-10"),
store: "Mumbai",
items: [
  { product_id: 101, qty: 2, price: 199 },
  { product_id: 102, qty: 1, price: 899 }
],
total: 199*2 + 899

},
{
order_id: "O1002",
customer: { id: 2, name: "Rina" },
date: new Date("2025-10-11"),
store: "Pune",
items: [
  { product_id: 103, qty: 1, price: 3200 }
],
total: 3200

},
{
order_id: "O1003",
customer: { id: 1, name: "Amit" },
```

```
date: new Date("2025-10-11"),
store: "Mumbai",
items: [
  { product_id: 201, qty: 1, price: 45000 },
  { product_id: 101, qty: 1, price: 199 }
],
total: 45000 + 199

},
{
  order_id: "O1004",
  customer: { id: 3, name: "Vikram" },
  date: new Date("2025-10-12"),
  store: "Delhi",
  items: [
    { product_id: 102, qty: 2, price: 899 }
  ],
  total: 899*2
}
]);
```

3. Indexing Basics (Create & Inspect)

a) Create a single-field index on orders.customer.id

```
db.orders.createIndex({ "customer.id": 1 }, { name: "idx_customer_id" })
```

b) Create a compound index for queries by store and date

```
db.orders.createIndex({ store: 1, date: -1 }, { name: "idx_store_date" })
```

c) Create an index on items.product_id (useful for queries that \$unwind then match)

```
db.orders.createIndex({ "items.product_id": 1 }, { name: "idx_items_product" })
```

d) Create a unique index (products._id already unique by default, but example:)

```
db.products.createIndex({ name: 1 }, { unique: true, name: "idx_product_name_unique" })
```

e) Create a text index for search on product name + category

```
db.products.createIndex({ name: "text", category: "text" }, { name: "idx_products_text" })
```

f) TTL index example (for session collection)

```
db.sessions.drop();
```

```
db.sessions.insertOne({ _id: "s1", user: "Amit", createdAt: new Date() });
```

```
db.sessions.createIndex({ createdAt: 1 }, { expireAfterSeconds: 60*60*24, name: "idx_sessions_ttl" });
```

(TTL index automatically removes documents older than the TTL.)

g) View indexes

```
db.orders.getIndexes();
```

```
db.products.getIndexes();
```

4. Check Index Usage with explain()

Run a query that should use the idx_customer_id index and view execution stats:

```
db.orders.find({ "customer.id": 1 }).explain("executionStats")
```

(Look for "indexName" : "idx_customer_id" and totalKeysExamined low -> index used.)

Now run a query that can benefit from the compound index:

```
db.orders.find({ store: "Mumbai", date: { $gte: new Date("2025-10-01") } })
  .sort({ date: -1 })
  .explain("executionStats")
```

(If index used, you'll see "indexName": "idx_store_date".)

5. Aggregation Pipeline Examples

All pipelines below are run against orders and products.

a) Total sales per store (group + sum)

```
db.orders.aggregate([
  { $group: { _id: "$store", totalSales: { $sum: "$total" }, orders: { $sum: 1 } },
  { $sort: { totalSales: -1 } }
])
```

b) Daily sales (group by date only)

```
db.orders.aggregate([
  { $group: { _id: { $dateToString: { format: "%Y-%m-%d", date: "$date" } },
    totalSales: { $sum: "$total" }, orders: { $sum: 1 } },
  { $sort: { _id: 1 } }
])
```

c) Top selling products (need to \$unwind items, group by product_id)

```
db.orders.aggregate([
  { $unwind: "$items" },
```

```

{ $group: { _id: "$items.product_id", qtySold: { $sum: "$items.qty" },
revenue: { $sum: { $multiply: ["$items.qty", "$items.price"] } } } },
{ $sort: { qtySold: -1 } },
{ $limit: 10 },
// join product details
{ $lookup: { from: "products", localField: "_id", foreignField: "_id", as:
"product" } },
{ $unwind: { path: "$product", preserveNullAndEmptyArrays: true } },
{ $project: { productId: "$_id", productName: "$product.name", category:
"$product.category", qtySold: 1, revenue: 1, _id: 0 } }
])

```

d) Customer-wise average order value

```

db.orders.aggregate([
{ $group: { _id: "$customer.id", customerName: { $first: "$customer.name" },
avgOrderValue: { $avg: "$total" }, orders: { $sum: 1 } } },
{ $sort: { avgOrderValue: -1 } }
])

```

e) Multi-facet analysis in one pass (\$facet)

```

db.orders.aggregate([
{
$facet: {
salesByStore: [
{ $group: { _id: "$store", totalSales: { $sum: "$total" } } },
{ $sort: { totalSales: -1 } }
]
}
])

```

```

    ],
  topProducts: [
    { $unwind: "$items" },
    { $group: { _id: "$items.product_id", qtySold: { $sum: "$items.qty" } } },
    { $sort: { qtySold: -1 } },
    { $limit: 5 },
    { $lookup: { from: "products", localField: "_id", foreignField: "_id", as: "prod" } },
    { $unwind: "$prod" },
    { $project: { productName: "$prod.name", qtySold: 1, _id: 0 } }
  ]
}
}
])

```

f) Bucketing orders by total amount (fixed buckets)

```
db.orders.aggregate([

```

```

{
  $bucket: {
    groupBy: "$total",
    boundaries: [0, 1000, 5000, 10000, 50000],
    default: "Other",
    output: {
      count: { $sum: 1 },

```

```

    orders: { $push: "$order_id" },
    totalRevenue: { $sum: "$total" }

}
}

}

])

```

g) Auto bucketing (Mongo chooses buckets)

```

db.orders.aggregate([
  { $bucketAuto: { groupBy: "$total", buckets: 3, output: { count: { $sum: 1 }, totalRevenue: { $sum: "$total" } } } }
])

```

h) Use \$lookup as a left join (orders → products) for each item (denormalize)
 (This is heavier because items is an array; typical approach: unwind then
 lookup)

```

db.orders.aggregate([
  { $unwind: "$items" },
  { $lookup: { from: "products", localField: "items.product_id", foreignField: "_id", as: "prod" } },
  { $unwind: { path: "$prod", preserveNullAndEmptyArrays: true } },
  { $project: { order_id: 1, customer: 1, store: 1, date: 1, "items.qty": 1, "items.price": 1, productName: "$prod.name", category: "$prod.category" } }
])

```

i) Pipeline with \$addFields and \$merge (save aggregated results to a new collection)

```

db.orders.aggregate([

```

```

{ $unwind: "$items" },
{ $group: { _id: "$items.product_id", qtySold: { $sum: "$items.qty" },
revenue: { $sum: { $multiply: ["$items.qty", "$items.price"] } } } },
{ $addFields: { lastUpdated: new Date() } },
{ $merge: { into: "product_sales", on: "_id", whenMatched: "replace",
whenNotMatched: "insert" } }
])

```

(After this, db.product_sales.find().pretty() contains aggregated product-level sales.)

6. Explain Aggregation Performance & Index Hints

Aggregation pipeline stages that can use indexes: early \$match and \$sort can benefit from indexes if they appear at start of pipeline.

Example: put \$match first so it can use an index:

```

db.orders.aggregate([
  { $match: { store: "Mumbai" } },           // uses idx_store_date index if present
  { $sort: { date: -1 } },
  { $limit: 50 }
]).explain("executionStats")

```

You can also use .hint() on aggregate() (Mongo 4.2+):

```

db.orders.aggregate([
  { $match: { store: "Mumbai" } },
  { $sort: { date: -1 } }
], { hint: { store: 1, date: -1 } }).explain("executionStats")

```

7. Text Search Example (Using Text Index)

```
// search products for 'mouse' or 'keyboard'  
db.products.find({ $text: { $search: "mouse keyboard" } }, { score: { $meta: "textScore" } }).sort({ score: { $meta: "textScore" } })
```

8. Index Maintenance & Viewing Stats

Show index usage stats:

```
db.products.aggregate([ { $indexStats: {} } ]).pretty()
```

```
db.orders.aggregate([ { $indexStats: {} } ]).pretty()
```

Drop an index:

```
db.orders.dropIndex("idx_store_date")
```

9. Best Practices & Tips

- Create indexes to support your most common and most selective queries. Indexes speed reads but slow writes and consume space.
- Put \$match and \$sort early in the pipeline so the server can use indexes.
- For large analytical pipelines, consider \$merge to write intermediate results to a collection and index those for repeated queries.
- Use compound indexes for queries that filter on multiple fields in a predictable order.
- Use TTL for ephemeral session-like data.
- Use text index for full-text search; beware only one text index per collection (but it can span multiple fields).
- Prefer set-based aggregation over client-side loops for performance.

10. Cleanup (Optional)

```
db.products.drop();
db.orders.drop();
db.product_sales.drop();
db.sessions.drop();
db.getMongo().close() // exit mongosh
```