MongoDB CRUD and Lookup Guide

## Introduction

MongoDB is a NoSQL database that stores data in BSON (Binary JSON) format. Unlike Oracle, which uses relational tables, MongoDB stores data as flexible JSON-like documents. This guide explains how to perform CRUD operations (Create, Read, Update, Delete) and lookups using the mongosh shell.

## Collection Overview

| **Collection Name** | **Document Count** | **Avg Doc Size (kB)** | **Relationship** |
| --- | --- | --- | --- |
| financialLevel1 | 8K | 3.13 |  |
| financialLevel2 | - | - | One-to-Many with financialLevel1 via longId → planId |
| financialLevel3 | 830K | 1.30 | One-to-Many with financialLevel1 via longId → ip\_long\_id, planId, costId |
| taxonomyMappings | 6M |  | One-to-Many with financialLevel1 via longId → ip\_long\_id |
| billingKeyHeader | 46K | 2.98 | One-to-Many with financialLevel1 via longId → IPLONGID |
| appMappings | 100K | 3.5 | One-to-One with financialLevel1 via longId → iplongid |

## Collection Schemas:

#### financialLevel1

| {  "\_id": "ObjectId",  "longId": "String",  "shortId": "String",  "title": "String",  "executionOnly": "String",  "openForTimeEntry": "String",  "type": "String",  "state": "String",  "benefitsReportingLevel": "String",  "productName": "String",  "startDate": "Date",  "endDate": "Date",  "owner": "String",  "agreementApprovers": "String",  "collaborators": "String",  "cto": "String",  "cbt": "String",  "sponsor": "String",  "primaryFbm": "String",  "additionalFbms": "String",  "owningOrganization": "String",  "inPlan": "String",  "openCreationDate": "Date",  "overview": "String",  "regionalImpact": "String",  "isMigrated": "Boolean",  "dataForEntitlementsCheck": {  "allowedWhiteSids": [  "String"  ]  },  "rolesPerPermissions": {  "read": [  "String"  ],  "write": [  "String"  ],  "admin": [  "String"  ]  },  "financials": {  "outlook2025": "Object"  } } |
| --- |

#### financialLevel3

| {  "\_id": "ObjectId",  "ip\_long\_id": "String",  "ip\_plan\_id": "String",  "ip\_cost\_id": "String",  "scenario": "String",  "year": "Number",  "mytech\_name": "String",  "location": "String",  "fy\_cost": "Number",  "monthCost": [  "Number"  ],  "monthHC": [  "Number"  ],  "currentMonth": "Number" } |
| --- |

#### billingKeyHeader

| {  "\_id": "ObjectId",  "IPLONGID": "String",  "PLANID": "String",  "HEADERID": "String",  "BKID": "String",  "REASON": "String",  "TYPE": "String",  "YEAR": "Number",  "Allocations": [  {  "SNODE": "Number",  "ALLOC\_PERCENT": "Number"  }  ] } |
| --- |

#### taxonomy

| {  "\_id": "ObjectId",  "ip\_long\_id": "String",  "year": "Number",  "scenario": "String",  "gt\_fin\_taxonomy\_l1": "String",  "gt\_fin\_taxonomy\_l2": "String",  "gt\_fin\_taxonomy\_name": "String",  "gt\_fin\_taxonomy\_percent": "Number" } |
| --- |

#### appMappings

| {  "\_id": "ObjectId",  "iplongid": "String",  "planid": "String",  "scenario": "String",  "softcapeligible": "String",  "year": "Number",  "months": [  {  "month": "String",  "applications": [  {  "appid": "String",  "apppct": "Number",  "cappct": "Number"  }  ]  }  ] } |
| --- |

## 

## 

## CRUD Operations

| **Insert** db.financialLevel1.insertOne({ longId: "12345", title: "Test Project" })  **Update** db.financialLevel1.updateOne({ longId: "12345" }, { $set: { title: "Updated Project" } } **Delete** db.financialLevel1.deleteOne({ longId: "12345" }) **Bulk Insert** db.financialLevel1.bulkWrite([{ insertOne: { document: { longId: "12348", title: "Project C" } } }]) **Bulk Update** db.financialLevel1.bulkWrite([{ updateOne: { filter: { longId: "12348" }, update: { $set: { status: "Closed" } } } }]) **Bulk Delete** db.financialLevel1.bulkWrite([{ deleteOne: { filter: { longId: "12348" } } }])  **UpdateMany** db.financialLevel3.updateMany([ { $set: { mnth\_cost: { $map: { input: { $split: ["$mnth\_cost", "|"] }, as: "value", in: { $toDouble: "$$value" } } }, mnth\_hc: { $map: { input: { $split: ["$mnth\_hc", "|"] }, as: "value", in: { $toDouble: "$$value" } } } } } ] ) |
| --- |

## Aggregation Operations:

Aggregation in MongoDB is used to process data and transform it into meaningful insights. Unlike normal queries (find()), which simply retrieve documents based on a filter, aggregation allows you to:

• Combine data from multiple collections

• Transform documents (reshape, filter, compute)

• Group data and calculate statistics

• Perform complex operations

### Common Aggregation Stages

| 1.$match - Filters documents based on criteria (similar to find())  2.$project - Reshapes the documents (includes/excludes fields, computes new fields)  3.$group - Groups documents and applies aggregation functions (e.g., $sum, $avg, $max, $min)  4.$lookup - Performs a left outer join with another collection (like JOIN in SQL) 5.$unwind - Deconstructs arrays (creates one document per element) 6.$sort - Sorts documents based on fields 7.$limit - Limits the number of documents processed 8.$skip - Skips a specified number of documents 9.$addFields - Adds new computed fields 10.$set - Alias for $addFields 11.$facet - Runs multiple aggregation pipelines within a single stage 12.$bucket - Categorizes documents into buckets (like GROUP BY) 13.$replaceRoot - Promotes a field to the top-level document 14.$out - Writes the result to a new collection 15.$merge - Writes the result back to an existing collection |
| --- |

## Indexing Strategies

| Create Index on longId:  db.financialLevel1.createIndex({ longId: 1 }) Compound Index on planId and costId:  db.financialLevel3.createIndex({ planId: 1, costId: 1 }) |
| --- |