
Capstone: Customer360 for DBS Tech Bank

1️⃣ Create Database

Command

```
use customer360_dbs
```

What it does

- Tells MongoDB: "Work inside a database called `customer360_dbs`".
- If the DB doesn't exist, MongoDB will create it when you first insert data.

Purpose

- Keeps all **Customer360-related collections** grouped in one logical place.
- Helps separate this project from other databases (like `test`, `admin`, etc.).
- Good practice for **clean data organization** in real projects.

2️⃣ Create Collections

Commands

```
db.createCollection("customers")
db.createCollection("accounts")
db.createCollection("transactions")
db.createCollection("alerts")
```

What they do

- Create **4 logical tables** (collections) in MongoDB:
  - `customers` → basic customer info
  - `accounts` → all bank accounts per customer
  - `transactions` → financial activity
  - `alerts` → messages/notifications (low balance, suspicious activity, etc.)

Purpose

- Mimics a **real banking system design**:
  - Separate collections keep data clean and modular.
  - Relationships are formed using `custId` and `accId`.
- Helps students understand how to **model relational-style data** in MongoDB (document store).

3️⃣ Insert Sample Data (Seed Dataset)

Here we create **realistic sample data** so that all future queries make sense.

👨 A. `customers` Collection

Command (example)

```
db.customers.insertMany([
  {
    custId: "C1001",
    name: "Amit Sharma",
    dob: "1993-04-12",
    gender: "Male",
    city: "Pune",
    mobile: "9876543210",
    email: "amit.sharma@example.com",
    kycStatus: "Verified"
  },
  {
    custId: "C1002",
    name: "Priya Iyer",
    dob: "1995-08-22",
    gender: "Female",
    city: "Mumbai",
    mobile: "9988776655",
    email: "priya.iyer@example.com",
    kycStatus: "Verified"
  }
])
```

Purpose

- Represents **who the customer is**:
  - Name, DOB, contact details, city, KYC.
- `custId` acts as a **business key** (like customer number in core banking).
- Customer360 always starts with **customer identity**.

🏦 B. `accounts` Collection

Command (example)

```
db.accounts.insertMany([
  {
    accId: "A2001",
    custId: "C1001",
    type: "Savings",
    branch: "Pune-Baner",
    balance: 85000
  },
  {
    accId: "A2002",
    custId: "C1001",
    type: "CreditCard",
    limit: 200000,
    dueAmount: 5000
  },
  {
    accId: "A2003",
    custId: "C1002",
    type: "Current",
    branch: "Mumbai-Andheri",
    balance: 250000
  }
])
```

Purpose

- Shows **what financial products** the customer holds:
  - Savings, Current, Credit Card, etc.
- Links back to `customers` using `custId`.
- Enables questions like:
  - "How many accounts does Amit have?"
  - "What is the total balance across all accounts?"

💳 C. `transactions` Collection

Command (example)

```
db.transactions.insertMany([
  {
    txnId: "T3001",
    accId: "A2001",
    amount: 5000,
    type: "Debit",
    mode: "UPI",
    timestamp: ISODate("2025-01-05T10:30:00Z")
  },
  {
    txnId: "T3002",
    accId: "A2001",
    amount: 15000,
    type: "Credit",
    mode: "Salary",
    timestamp: ISODate("2025-01-01T09:10:00Z")
  }
])
```

Purpose

- Tracks **money movement**:
  - Credits (inflow) and Debits (outflow).
- Links to `accounts` using `accId`.
- Enables:
  - Monthly statements,
  - Spending analysis,
  - Fraud/risk monitoring,
  - "Last N transactions" view.

🚨 D. `alerts` Collection (Optional)

Command (example)

```
db.alerts.insertOne({
  alertId: "AL4001",
  custId: "C1001",
  type: "BalanceLow",
  message: "Your balance dropped below ₹10,000",
  timestamp: ISODate("2025-01-12T15:00:00Z")
})
```

Purpose

- Simulates **real-time alerts/notifications** sent by the bank.
- Links directly to `customers` using `custId`.
- Valuable for the 360° view:

- Show latest alerts together with transactions & accounts.

4️⃣ Build Customer360 Aggregations

Now we use **aggregation pipelines** to combine data across collections and generate insights.

⭐ Task 1: Full Customer360 Profile

Pipeline

1. **`$match`** – filter by customer
2. `{ $match: { custId: "C1001" } }`
   - Choose just **one customer** to build the 360° profile.
   - Reduces data early → better performance.

```
db.customers.aggregate([
  { $match: { custId: "C1001" } },

  {
    $lookup: {
      from: "accounts",
      localField: "custId",
      foreignField: "custId",
      as: "accounts"
    }
  },

  {
    $lookup: {
      from: "transactions",
      localField: "accounts.accId",
      foreignField: "accId",
      as: "transactions"
    }
  },

  {
    $lookup: {
      from: "alerts",
      localField: "custId",
      foreignField: "custId",
      as: "alerts"
    }
  },

  {
```

```
    $addFields: {
      totalBalance: { $sum: "$accounts.balance" },
      last5Transactions: {
        $slice: [
          { $sortArray: { input: "$transactions", sortBy: {
timestamp: -1 } } },
          5
        ]
      },
      latestAlert: {
        $arrayElemAt: [
          { $sortArray: { input: "$alerts", sortBy: {
timestamp: -1 } } },
          0
        ]
      }
    }
  },

  {
    $project: {
      _id: 0,
      custId: 1,
      name: 1,
      city: 1,
      mobile: 1,
      email: 1,
      accounts: 1,
      totalBalance: 1,
      last5Transactions: 1,
      latestAlert: 1
    }
  }
])
```

Step-by-step Purpose

3. **`$match`** – filter by customer
4. `{ $match: { custId: "C1001" } }`
   - Choose just **one customer** to build the 360° profile.
   - Reduces data early → better performance.
5. **`$lookup` with `accounts`**
   - Joins `customers` with `accounts` using `custId`.
   - Adds an array field `accounts` to the customer document.
   - Purpose: see **all products** this customer has.
6. **`$lookup` with `transactions`**
   - Joins using `accounts.accId` → `transactions.accId`.
   - Collects all transactions for all accounts into `transactions` array.
   - Purpose: attach **full transaction history** under the same customer.
7. **`$lookup` with `alerts`**

- Joins alerts by `custId`.
- Adds an `alerts` array.
- Purpose: bring in **communication/alerts** as part of 360°.

8.
9. **`$addFields`**
   - `totalBalance`: sum of all `accounts.balance`.
   - `last5Transactions`: sort transactions by timestamp and slice top 5.
   - `latestAlert`: get the most recent alert.
   - Purpose:
     - Convert **raw data → meaningful metrics**:
       - "How much total money?"
       - "What are the latest activities?"
       - "What is the last message we sent?"
10. **`$project`**
    - Selects which fields should appear in the final result.
    - Hides internal fields like `_id`.
    - Purpose: produce a **clean, API-friendly Customer360 JSON**.

⭐ Task 2: Total Credits & Debits Per Customer

Pipeline

```
db.transactions.aggregate([
  {
    $lookup: {
      from: "accounts",
      localField: "accId",
      foreignField: "accId",
      as: "account"
    }
  },
  { $unwind: "$account" },

  {
    $group: {
      _id: "$account.custId",
      totalCredits: {
        $sum: { $cond: [ { $eq: ["$type", "Credit"] },
"$amount", 0 ] }
      },
      totalDebits: {
        $sum: { $cond: [ { $eq: ["$type", "Debit"] },
"$amount", 0 ] }
      }
    }
  }
])
```

Purpose of steps

- `$lookup`:
  - Join each transaction with its account (`accId`).
  - Needed to know **which customer** the transaction belongs to.
- `$unwind`:
  - Convert the `account` array into a single object to simplify grouping.
- `$group` by `custId`:
  - For each customer, sum:
    - All **credit amounts**.
    - All **debit amounts**.
  - Purpose: get **financial behavior** per customer – total inflow vs outflow.

⭐ Task 3: Monthly Statement for an Account

Pipeline

```
db.transactions.aggregate([
  { $match: { accId: "A2001" } },

  {
    $group: {
      _id: { month: { $month: "$timestamp" }, year: { $year:
"$timestamp" } },
      totalCredits: {
        $sum: { $cond: [ { $eq: ["$type", "Credit"] },
"$amount", 0 ] }
      },
      totalDebits: {
        $sum: { $cond: [ { $eq: ["$type", "Debit"] },
"$amount", 0 ] }
      },
      transactions: { $push: "$$ROOT" }
    }
  }
])
```

Purpose

- `$match`: focus on one account (like generating bank statement).
- `$group` by month + year:
  - Group all transactions by month/year.
- `totalCredits` & `totalDebits`: summarised totals per month.
- `transactions: { $push: "$$ROOT" }`:
  - Include raw transaction details in the output.
- Overall purpose: simulate a **bank account statement feature**.

⭐ Task 4: Top 5 High-Value Customers

Pipeline

```
db.accounts.aggregate([
  {
    $group: {
      _id: "$custId",
      totalBalance: { $sum: "$balance" }
    }
  },
  { $sort: { totalBalance: -1 } },
  { $limit: 5 }
])
```

Purpose

- `$group` by `custId`: calculate **total balance per customer**.
- `$sort` descending by `totalBalance`: biggest balances first.
- `$limit` 5: only top 5.
- Helps business teams identify **priority customers** for offers, relationship management, etc.

⭐ Task 5: Customers with No Recent Transactions

Pipeline

```
db.transactions.aggregate([
  {
    $group: {
      _id: "$accId",
      lastTxn: { $max: "$timestamp" }
    }
  },
  {
    $match: {
      lastTxn: { $lt: ISODate("2025-01-01") }
    }
  }
])
```

Purpose

- `$group` by `accId`: find **last transaction date** per account.
- `$match` on `lastTxn`: accounts where last transaction is **older than a cutoff date**.
- Use case:

- Identify **dormant / inactive accounts**.
- Trigger re-engagement campaigns, compliance checks, etc.

5️⃣ Indexing Tasks – Purpose

A. Index on `custId`

```
db.customers.createIndex({ custId: 1 }, { unique: true })
```

Purpose

- Faster lookup by `custId` (used in many queries).
- Enforces **uniqueness** → no duplicate customer IDs.

B. Index on `accId`

```
db.accounts.createIndex({ accId: 1 }, { unique: true })
```

Purpose

- Same idea: quick searches by `accId`.
- Critical for joins (`$lookup`) and direct lookups.

C. Compound Index on `accId` + `timestamp`

```
db.transactions.createIndex({ accId: 1, timestamp: -1 })
```

Purpose

- Optimizes queries that:
  - Filter by account (`accId`) and
  - Sort by transaction date (`timestamp`).
- Perfect for:
  - Recent transactions,
  - Statements,
  - "Last N transaction" views.

D. Explain Query Plan

```
db.customers.find({ custId: "C1001"
}).explain("executionStats")
```

Purpose

- Shows how MongoDB executes the query.
- Helps verify:
  - Is the **index being used**?
  - How many documents were scanned?
- Important for **performance tuning** in enterprise setups.
