Capstone Project: Customer360 Database for DBS Tech Bank

🎯 Objective

Design and implement a **MongoDB-based Customer360 system** for DBS Tech Bank with:

- Collections: **customers, accounts, transactions, alerts**
- Proper **data model**, **indexes**, and **relationships**
- **Document-based ER diagram**
- **Sample queries** to power Customer360 dashboards, statements, and alerts

---

Step 0: Project Setup

What you do

1. Open mongosh and select the project DB:

```
use dbs_tech_bank;
```

2. (Optional) Create collections explicitly:

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

What to verify

- Run show collections and confirm all four collections exist.

Step 1: Understand Customer360 Requirements

Customer360 = **single view of the customer** across products & activity.

Key business questions

1. Who is the customer? (profile, KYC, risk)
2. What accounts do they hold? (savings, current, loans, cards)
3. What is their recent activity? (last N transactions)
4. Are there any alerts or fraud flags?
5. What is their overall relationship value? (balances, spend)

What you do

Write down 5–10 **queries/features** you want to support, for example:

- “Show full profile + accounts + last 5 transactions for custId = C1001”
- “Find high-value customers by total monthly debit”
- “List all alerts raised for a customer in last 30 days”

This drives your **schema design & indexes**.

Step 2: Design Collections & Key Fields

2.1 customers collection

Purpose: Master record of each customer.

```
{
  "_id": ObjectId("..."),
  "custId": "C1001",                      // business key (unique)
  "fullName": "Rohit Sharma",
  "dob": "1988-04-15",
  "email": "rohit.sharma@example.com",
  "mobile": "9876543210",
  "kycStatus": "VERIFIED",                // PENDING / VERIFIED / REJECTED
  "riskRating": "LOW",                    // LOW / MEDIUM / HIGH
  "addresses": [
    {
      "type": "home",
      "line1": "Pune Nagar Road",
      "city": "Pune",
      "state": "MH",
      "pin": "411014",
      "country": "IN"
    }
  ],
  "createdAt": ISODate("2024-01-10T10:00:00Z"),
  "updatedAt": ISODate("2025-01-10T10:00:00Z")
}
```

2.2 accounts collection

Purpose: Bank accounts owned by customers.

```
{
  "_id": ObjectId("..."),
  "accNo": "SAV1001",                     // business key (unique)
  "custId": "C1001",                      // link to customers.custId
  "productCode": "SAVINGS_STD",           // link to product catalog (future)
  "type": "SAVINGS",                      // SAVINGS / CURRENT / LOAN / CARD
  "currency": "INR",
  "branchCode": "BR_PUNE_01",
  "status": "ACTIVE",                     // ACTIVE / CLOSED
  "balance": 152340.75,
  "openedAt": ISODate("2023-10-10T09:00:00Z")
}
```

2.3 transactions collection

Purpose: All financial activity on accounts.

```
{
  "_id": ObjectId("..."),
  "txnId": "T9001",                    // transaction reference
  "accNo": "SAV1001",                  // link to accounts.accNo
  "custId": "C1001",                   // denormalized for faster filters
  "type": "DEBIT",                     // DEBIT / CREDIT
  "amount": 2000.00,
  "currency": "INR",
  "channel": "UPI",                    // ATM / UPI / NEFT / RTGS / POS
  "location": "Pune",
  "timestamp": ISODate("2025-01-02T14:45:00Z"),
  "status": "POSTED"                   // POSTED / PENDING / REVERSED
}
```

2.4 alerts collection

Purpose: Store fraud/compliance alerts raised from rules.

```
{
  "_id": ObjectId("..."),
  "alertId": "A5001",
  "custId": "C1003",
  "accNo": "CURR1003",
  "alertType": "HIGH_DAILY_DEBIT",     // rule code
  "severity": "HIGH",                  // LOW / MEDIUM / HIGH / CRITICAL
  "description": "Total daily debit exceeded ₹10,00,000.",
  "ruleDetails": {
    "thresholdAmount": 1000000,
    "actualAmount": 1580000,
    "txnCount": 5,
    "date": "2025-01-10"
  },
  "status": "OPEN",                    // OPEN / IN_PROGRESS / CLOSED
  "createdAt": ISODate("2025-01-10T13:00:00Z"),
  "updatedAt": ISODate("2025-01-10T13:30:00Z")
}
```

What you do

- Write the **JSON structure** for each collection (like above).
- Decide which fields are **mandatory** vs **optional**.

Step 3: Define Relationships (Document-Based ER Diagram)

Relationships

- **customers – accounts**: 1 → many
- **accounts – transactions**: 1 → many
- **customers – alerts**: 1 → many

- **accounts – alerts**: 1 → many (optional, depending on alert)

ASCII ER (document-style) Diagram

```
+------------+        +------------+        +----------------+
| customers  | 1    * |  accounts  | 1    * |  transactions  |
+------------+        +------------+        +----------------+
| custId PK  |<-------| custId FK  |        | accNo FK       |
| fullName   |        | accNo PK   |<-------| custId (denorm)
| mobile     |        | balance    |        | amount         |
+------------+        +------------+        +----------------+
       |
       | 1    *
       v
+------------+
|  alerts    |
+------------+
| alertId PK |
| custId FK  |
| accNo FK   |
| alertType  |
+------------+
```

Where:

- **PK** = primary business key
- **FK** = foreign key / reference field

What you do

- Draw this diagram neatly (in draw.io / PowerPoint / paper).
- Label **1–many** relationships and key fields.
- Include this as **Capstone deliverable**.

Step 4: Indexing Strategy

4.1 customers indexes

```
db.customers.createIndex({ custId: 1 }, { unique: true });
db.customers.createIndex({ mobile: 1 }, { unique: true });
db.customers.createIndex({ email: 1 }, { unique: true });
db.customers.createIndex({ kycStatus: 1, riskRating: 1 });
```

Why:

- Fast lookup by **custId**, **mobile**, **email**

- Dashboards on **KYC** and **Risk**

4.2 accounts indexes

```
db.accounts.createIndex({ accNo: 1 }, { unique: true });
db.accounts.createIndex({ custId: 1, status: 1 });
```

Why:

- Fast lookup by account number
- List all **active** accounts for a customer

4.3 transactions

indexes

```
// For account statements
db.transactions.createIndex({ accNo: 1, timestamp: -1 });

// For customer-level analytics
db.transactions.createIndex({ custId: 1, timestamp: -1 });

// For fraud / channel analytics
db.transactions.createIndex({ channel: 1, timestamp: -1 });
```

4.4 alerts indexes

```
db.alerts.createIndex({ custId: 1, createdAt: -1 });
db.alerts.createIndex({ status: 1, severity: -1 });
```

What you do

- Implement all createIndex commands.
- Run db.collection.getIndexes() to verify.

Step 5: Implement Schema & Load Sample Data

What you do

1. Insert **5–10 customers** into customers.
2. Insert **2–3 accounts per customer** into accounts.
3. Insert **20–50 transactions** per account into transactions.
4. Insert **a few alerts** manually or via aggregation pipeline.

You can reuse insertion patterns from earlier labs.

Step 6: Sample Queries (Must Include in Capstone)

6.1 Basic – Get Customer Profile

```
db.customers.find(
  { custId: "C1001" },
  { _id: 0, custId: 1, fullName: 1, mobile: 1, email: 1, kycStatus: 1,
riskRating: 1 }
);
```

6.2 List All Accounts of a Customer

```
db.accounts.find(
  { custId: "C1001", status: "ACTIVE" },
  { _id: 0, accNo: 1, type: 1, balance: 1, currency: 1 }
);
```

6.3 Get Last 5 Transactions of an Account

```
db.transactions.find(
  { accNo: "SAV1001", status: "POSTED" },
  { _id: 0, txnId: 1, type: 1, amount: 1, channel: 1, timestamp: 1 }
)
.sort({ timestamp: -1 })
.limit(5);
```

6.4 Customer360 View – Combined (Aggregation)

Goal: For a given custId, show:

- Profile (from customers)
- Accounts summary (count & total balance)
- Last 5 transactions (across all accounts)

```
db.customers.aggregate([
  // 1. Filter customer
  { $match: { custId: "C1001" } },

  // 2. Lookup accounts
  {
    $lookup: {
      from: "accounts",
      localField: "custId",
      foreignField: "custId",
      as: "accounts"
    }
  },

  // 3. Lookup last 5 transactions across all accounts
  {
    $lookup: {
      from: "transactions",
      let: { customerId: "$custId" },
      pipeline: [
        { $match: { $expr: { $eq: ["$custId", "$$customerId"] } } },
        { $sort: { timestamp: -1 } },
        { $limit: 5 },
        {
          $project: {
            _id: 0,
            txnId: 1,
            accNo: 1,
            type: 1,
            amount: 1,
            channel: 1,
            timestamp: 1
          }
        }
      ],
      as: "recentTransactions"
    }
  },

  // 4. Add account summary
  {
    $addFields: {
      totalAccounts: { $size: "$accounts" },
      totalBalance: { $sum: "$accounts.balance" }
    }
  },

  // 5. Final projection
  {
    $project: {
      _id: 0,
      custId: 1,
      fullName: 1,
      mobile: 1,
      kycStatus: 1,
      riskRating: 1,
      totalAccounts: 1,
      totalBalance: 1,
      accounts: {
        accNo: 1,
        type: 1,
        balance: 1,
        status: 1
      },
      recentTransactions: 1
    }
  }
]);
```

6.5 Monthly Summary per Customer (for Analytics)

```
db.transactions.aggregate([
  {
    $match: {
      custId: "C1001",
      status: "POSTED"
    }
  },
  {
    $group: {
      _id: {
        year: { $year: "$timestamp" },
        month: { $month: "$timestamp" }
      },
      totalDebit: {
        $sum: {
          $cond: [{ $eq: ["$type", "DEBIT"] }, "$amount", 0]
        }
      },
      totalCredit: {
        $sum: {
          $cond: [{ $eq: ["$type", "CREDIT"] }, "$amount", 0]
        }
      },
      txnCount: { $sum: 1 }
    }
  },
  {
    $project: {
      _id: 0,
      year: "$_id.year",
      month: "$_id.month",
      totalDebit: 1,
      totalCredit: 1,
      txnCount: 1
    }
  },
  { $sort: { year: 1, month: 1 } }
]);
```

6.6 Show Alerts for a Customer

```
db.alerts.find(
  { custId: "C1003" },
  { _id: 0, alertId: 1, alertType: 1, severity: 1, status: 1, createdAt: 1
}
)
.sort({ createdAt: -1 });
```

Step 7: Fraud Alert Generation (Optional Bonus)

Use an aggregation pipeline (like in your Lab 2) to **detect suspicious patterns** and insert into alerts:

- Group by custId + date
- Where totalDailyDebit > 10,00,000
- Insert a document in alerts for each suspicious day.

This shows **end-to-end flow**: transactions → analytics → alerts.

Step 8: Capstone Deliverables

1. **Project Report (2–4 pages)**
    - Problem statement
    - Collections & field descriptions
    - Relationships explanation
    - Indexing strategy and justification
2. **Document-based ER Diagram**
    - Clean diagram (image or PPT slide)
3. **MongoDB Script File(s)**
    - create_collections_and_indexes.js
    - insert_sample_data.js
4. **Sample Query Script**
    - All key queries listed above (Customer360, monthly summary, alerts)
5. (Optional) **Short Demo Video / Screenshots**
    - Running key queries in mongosh or MongoDB Compass.