Lab 3: Implement Account Transaction Logging using MongoDB

Learning Outcomes

By the end of this lab, participants will be able to:

1. Design and create a **transactions** collection
2. Insert realistic **banking transactions** with timestamps
3. Query **high-value transactions** using $gt
4. Use aggregation & $sum to compute **totals**
5. Build a **fraud-detection pipeline** using the aggregation framework

---

🧩 Step 0 – Switch to the Training Database

```
use dbs_tech_bank;
```

Explanation:

This selects the **DBS Tech Bank** database. If it doesn't exist yet, MongoDB will create it once we insert data.

1️⃣ Step 1 – Create the transactions Collection

You can let MongoDB create the collection automatically on first insert, but for clarity we'll create it explicitly.

```
db.createCollection("transactions");
```

Explanation:

- Defines a new collection named transactions.
- Good for teaching because it makes the structure explicit.

✅ Verify:

```
show collections;
```

You should see transactions in the list.

2️⃣ Step 2 – Insert Transaction Logs (with timestamps)

We'll insert realistic transactions for **3 customers** and **3 accounts**.

2.1 Insert Sample Transactions

```
db.transactions.insertMany([
  // Customer C1001 – Pune – normal + a high-value txn
  {
    txnId: "T2001",
    custId: "C1001",
    accNo: "SAV1001",
    type: "DEBIT",
    amount: 2500,
    channel: "UPI",
    location: "Pune",
    timestamp: ISODate("2025-01-10T09:15:00Z"),
    status: "POSTED"
  },
  {
    txnId: "T2002",
    custId: "C1001",
    accNo: "SAV1001",
    type: "DEBIT",
    amount: 90000,
    channel: "NETBANKING",
    location: "Pune",
    timestamp: ISODate("2025-01-10T11:45:00Z"),
    status: "POSTED"
  },

  // Customer C1002 – Mumbai – moderate usage
  {
    txnId: "T2003",
    custId: "C1002",
    accNo: "SAV1002",
    type: "CREDIT",
    amount: 50000,
    channel: "NEFT",
    location: "Mumbai",
    timestamp: ISODate("2025-01-10T10:00:00Z"),
    status: "POSTED"
  },
  {
    txnId: "T2004",
    custId: "C1002",
    accNo: "SAV1002",
    type: "DEBIT",
    amount: 20000,
    channel: "ATM",
    location: "Mumbai",
```

```
    timestamp: ISODate("2025-01-10T13:30:00Z"),
    status: "POSTED"
  },

  // Customer C1003 – Bangalore – suspicious pattern (multiple
huge debits same day)
  {
    txnId: "T2005",
    custId: "C1003",
    accNo: "CURR1003",
    type: "DEBIT",
    amount: 500000,
    channel: "NETBANKING",
    location: "Bangalore",
    timestamp: ISODate("2025-01-10T12:00:00Z"),
    status: "POSTED"
  },
  {
    txnId: "T2006",
    custId: "C1003",
    accNo: "CURR1003",
    type: "DEBIT",
    amount: 480000,
    channel: "NETBANKING",
    location: "Bangalore",
    timestamp: ISODate("2025-01-10T12:10:00Z"),
    status: "POSTED"
  },
  {
    txnId: "T2007",
    custId: "C1003",
    accNo: "CURR1003",
    type: "DEBIT",
    amount: 600000,
    channel: "NETBANKING",
    location: "Delhi",
    timestamp: ISODate("2025-01-10T12:20:00Z"),
    status: "POSTED"
  },
  {
    txnId: "T2008",
    custId: "C1003",
    accNo: "CURR1003",
    type: "CREDIT",
    amount: 800000,
    channel: "NEFT",
    location: "Bangalore",
    timestamp: ISODate("2025-01-11T09:30:00Z"),
    status: "POSTED"
  }
]);
```

Explanation:

- We're simulating **real banking behaviour**.
- C1003 is deliberately given **multiple huge debits in a short window** → good for fraud-detection demo.
- Each record has:
  - txnId, custId, accNo
  - type, amount, channel, location
  - timestamp and status

2.2 Verify the Insert

```
db.transactions.find().pretty();
```

Explanation:

- Ensures data is present and structured as expected.
- Learners visually connect the JSON structure to business meaning.

3️⃣ Step 3 – Query High-Value Transactions ($gt)

3.1 Find All Transactions > ₹1,00,000

```
db.transactions.find(
  { amount: { $gt: 100000 } },
  { _id: 0, txnId: 1, custId: 1, accNo: 1, amount: 1, channel:
1, timestamp: 1 }
);
```

Explanation:

- { amount: { $gt: 100000 } }
  - $gt = *greater than*
  - Filters transactions with amount > 100000.
- Projection { _id: 0, ... } hides _id and shows only key fields.

👉 Ask learners: *Which customer looks risky?*

Answer: C1003.

3.2 High-Value DEBIT Transactions Only

```
db.transactions.find(
  { type: "DEBIT", amount: { $gt: 100000 } },
  { _id: 0, txnId: 1, custId: 1, accNo: 1, amount: 1, channel:
1, timestamp: 1 }
```

```
);
```

Explanation:

- Adds type: "DEBIT" → focusing on **money going out**, more relevant for fraud.
- Real-world: high-value debits are reviewed more seriously than credits.

4️⃣ Step 4 – Use $sum to Compute Totals

Now we move from **row-level** queries to **aggregated summaries** using the **aggregation pipeline**.

4.1 Total Transaction Amount per Customer (All Types)

```
db.transactions.aggregate([
  {
    $group: {
      _id: "$custId",
      totalAmount: { $sum: "$amount" },
      txnCount: { $sum: 1 }
    }
  },
  {
    $project: {
      _id: 0,
      custId: "$_id",
      totalAmount: 1,
      txnCount: 1
    }
  }
]);
```

Explanation (stage-by-stage):

1. $group
   - _id: "$custId" → group by customer
   - totalAmount: { $sum: "$amount" } → adds all amounts per customer
   - txnCount: { $sum: 1 } → increments count for each txn
2. $project
   - Renames _id → custId
   - Keeps totalAmount, txnCount

Outcome:

You see **total volume** and **number of transactions** per customer.

4.2 Separate Total Debit and Total Credit per Customer

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

Explanation:

- $cond works like **IF condition** inside aggregation.
- For DEBIT:
  - If type == "DEBIT" → add amount
  - Else → add 0
- Same logic for CREDIT.

Teaching point:

This is how we derive **net outflow** and **inflow** for each customer.

5️⃣ Step 5 – Build a Fraud-Detection Query Pipeline

Now the main part of Lab 3: **fraud detection** using aggregation.

Fraud Rule (for this lab)

"Raise suspicion if a customer's **total DEBIT amount on a single day** exceeds **₹10,00,000**."

We'll compute **daily debit totals per customer** and filter by a threshold.

5.1 Add a txnDateOnly

Field and Group

We use $dateToString to convert timestamp to "YYYY-MM-DD".

```
db.transactions.aggregate([
  // 1. Consider only POSTED DEBIT transactions
  {
    $match: {
      status: "POSTED",
      type: "DEBIT"
    }
  },

  // 2. Derive a date-only field
  {
    $addFields: {
      txnDateOnly: {
        $dateToString: { format: "%Y-%m-%d", date:
"$timestamp" }
      }
    }
  },

  // 3. Group by customer + day
  {
    $group: {
      _id: {
        custId: "$custId",
        txnDateOnly: "$txnDateOnly"
      },
      totalDailyDebit: { $sum: "$amount" },
      txnCount: { $sum: 1 },
      txns: {
        $push: {
          txnId: "$txnId",
          accNo: "$accNo",
          amount: "$amount",
```

```
          channel: "$channel",
          location: "$location",
          timestamp: "$timestamp"
        }
      }
    }
  },

  // 4. Reshape output
  {
    $project: {
      _id: 0,
      custId: "$_id.custId",
      txnDateOnly: "$_id.txnDateOnly",
      totalDailyDebit: 1,
      txnCount: 1,
      txns: 1
    }
  }
]).pretty();
```

Explanation (stage-by-stage):

1. $match
   - Keep only **DEBIT + POSTED** txns → focus on outgoing confirmed money.
2. $addFields
   - txnDateOnly converts full timestamp to a date string like "2025-01-10".
3. $group
   - _id.custId + _id.txnDateOnly → one row per **customer per day**.
   - totalDailyDebit = sum of amounts for that day.
   - txnCount = how many txns in that day.
   - txns: { $push: { ... } } → collects detailed txns in an array (useful for investigation screen).
4. $project
   - Cleans up fields for output.

👉 At this point, you have daily debit totals. Next, you apply your **fraud rule**.

5.2 Apply Fraud Threshold ($match on aggregated values)

Add one more stage to filter suspicious days:

```
db.transactions.aggregate([
  {
    $match: {
      status: "POSTED",
```

```
      type: "DEBIT"
    }
  },
  {
    $addFields: {
      txnDateOnly: {
        $dateToString: { format: "%Y-%m-%d", date:
"$timestamp" }
      }
    }
  },
  {
    $group: {
      _id: {
        custId: "$custId",
        txnDateOnly: "$txnDateOnly"
      },
      totalDailyDebit: { $sum: "$amount" },
      txnCount: { $sum: 1 },
      txns: {
        $push: {
          txnId: "$txnId",
          accNo: "$accNo",
          amount: "$amount",
          channel: "$channel",
          location: "$location",
          timestamp: "$timestamp"
        }
      }
    }
  },
  {
    $project: {
      _id: 0,
      custId: "$_id.custId",
      txnDateOnly: "$_id.txnDateOnly",
      totalDailyDebit: 1,
      txnCount: 1,
      txns: 1
    }
  },
  // 5. Fraud rule: daily debit > 10,00,000
  {
    $match: {
      totalDailyDebit: { $gt: 1000000 }
    }
  },
  // 6. Sort by highest daily debit
  {
    $sort: { totalDailyDebit: -1 }
  }
```

```
]).pretty();
```

Explanation:

- Final $match filters only those **customer + day** combos where totalDailyDebit > 1000000.
- $sort orders suspicious cases by severity.
- With our sample data, **customer C1003 on 2025-01-10** should appear with a very high totalDailyDebit.

This is your **fraud detection query pipeline**.

6️⃣ Step 6 – (Optional) Prepare Alerts for an alerts Collection

If you want to go one more step and connect Lab 3 with your Customer360 DB:

6.1 Insert Suspicious Results into alerts

First, create the alerts collection if not present:

```
db.createCollection("alerts");
```

Then, manually insert an alert based on the fraud output for C1003:

```
db.alerts.insertOne({
  alertId: "A9001",
  custId: "C1003",
  accNo: "CURR1003",
  alertType: "HIGH_DAILY_DEBIT",
  severity: "HIGH",
  description: "Total daily debit exceeded ₹10,00,000 on 2025-
01-10.",
  ruleDetails: {
    thresholdAmount: 1000000,
    actualAmount: 1580000,
    txnDateOnly: "2025-01-10"
  },
  status: "OPEN",
  createdAt: ISODate("2025-01-10T13:00:00Z")
});
```

Explanation:

- Demonstrates how **analytics output (aggregation)** feeds into **operational collections** like alerts.
- This connects **data engineering** with **risk operations**.

✅ Lab 3 – Wrap-up Checklist

By the end of this hands-on, learners should have:

- Created a transactions collection
- Inserted realistic banking transactions
- Queried high-value transactions using $gt
- Used $sum inside $group to compute totals
- Built an **aggregation-based fraud detection pipeline** using:
    - $match
    - $addFields
    - $group
    - $project
    - $sort