Module_7_Capstone – Customer360 Database for DBS Tech Bank

🎯 Capstone Goal

Design and implement a **Customer360 Database** for **DBS Tech Bank** using **MongoDB**, and map everything to **equivalent SQL** so that learners can think in both worlds.

Scope

- Collections / Tables:
  - customers
  - accounts
  - transactions
  - alerts
- Include features from:
  - **Lab 1** – Data model, document design, basic CRUD & joins ($lookup)
  - **Lab 2** – Account Transaction Logging & summaries
  - **Lab 3** – Fraud Detection Pipeline
- Deliverables:
  - Document-based **ER diagram**
  - **Key fields & indexing strategy**
  - **Sample MongoDB queries** + **equivalent SQL**

A. Schema Design – Collections & Tables

1. customers – Customer Master

🟢 MongoDB: Document Shape

```
// Collection: customers
{
  _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")
}
```

🔵 SQL: Table Definition

```
CREATE TABLE customers (
    cust_id     VARCHAR(20) PRIMARY KEY,
    full_name   VARCHAR(100),
    dob         DATE,
    email       VARCHAR(100) UNIQUE,
    mobile      VARCHAR(20)  UNIQUE,
    kyc_status  VARCHAR(20),
    risk_rating VARCHAR(20),
    created_at  TIMESTAMP,
    updated_at  TIMESTAMP
);

/* addresses would typically be a separate table in SQL */
CREATE TABLE customer_addresses (
    addr_id   BIGINT PRIMARY KEY,
    cust_id   VARCHAR(20) REFERENCES customers(cust_id),
    addr_type VARCHAR(20),
    line1     VARCHAR(200),
    city      VARCHAR(50),
    state     VARCHAR(50),
    pin       VARCHAR(10),
    country   VARCHAR(10)
);
```

---

2. accounts – Customer Accounts

🟢 MongoDB

```
// Collection: accounts
{
  _id: ObjectId("..."),
  accNo: "SAV1001",                    // unique account number
  custId: "C1001",                     // link to customers.custId
  productCode: "SAV_STD",
  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")
}
```

🔵 SQL

```
CREATE TABLE accounts (
    acc_no       VARCHAR(20) PRIMARY KEY,
    cust_id      VARCHAR(20) REFERENCES customers(cust_id),
    product_code VARCHAR(50),
    type         VARCHAR(20),
    currency     VARCHAR(10),
    branch_code  VARCHAR(20),
    status       VARCHAR(20),
    balance      DECIMAL(18,2),
    opened_at    TIMESTAMP
);
```

---

3. transactions – Account Transactions (Day2: Lab2 + Lab3)

🟢 MongoDB

```
// Collection: transactions
{
  _id: ObjectId("..."),
  txnId: "T3001",
  custId: "C1001",                    // denormalized for faster filters
  accNo: "SAV1001",                   // link to accounts.accNo
  type: "DEBIT",                      // DEBIT / CREDIT
  amount: 90000,
  channel: "NETBANKING",              // UPI / ATM / NEFT / etc.
  location: "Pune",
  timestamp: ISODate("2025-01-10T11:45:00Z"),
  status: "POSTED"                    // POSTED / PENDING / REVERSED
}
```

🔵 SQL

```
CREATE TABLE transactions (
    txn_id    VARCHAR(20) PRIMARY KEY,
    cust_id   VARCHAR(20) REFERENCES customers(cust_id),
    acc_no    VARCHAR(20) REFERENCES accounts(acc_no),
    type      VARCHAR(10),      -- DEBIT / CREDIT
    amount    DECIMAL(18,2),
    channel   VARCHAR(20),
    location  VARCHAR(50),
    ts        TIMESTAMP,
    status    VARCHAR(20)
);
```

---

4. alerts – Fraud / Compliance Alerts (Day2: Lab3)

🟢 MongoDB

```
// Collection: alerts
{
```

```
  _id: ObjectId("..."),
  alertId: "A9001",
  custId: "C1003",
  accNo: "CURR1003",
  alertType: "HIGH_DAILY_DEBIT",
  severity: "HIGH",                      // LOW / MEDIUM / HIGH / CRITICAL
  description: "Total daily debit exceeded ₹10,00,000 on 2025-01-10.",
  ruleDetails: {
    thresholdAmount: 1000000,
    actualAmount: 1580000,
    txnCount: 3,
    txnDateOnly: "2025-01-10"
  },
  status: "OPEN",                        // OPEN / IN_PROGRESS / CLOSED
  createdAt: ISODate("2025-01-10T13:00:00Z"),
  updatedAt: ISODate("2025-01-10T13:30:00Z")
}
```

🔵 SQL

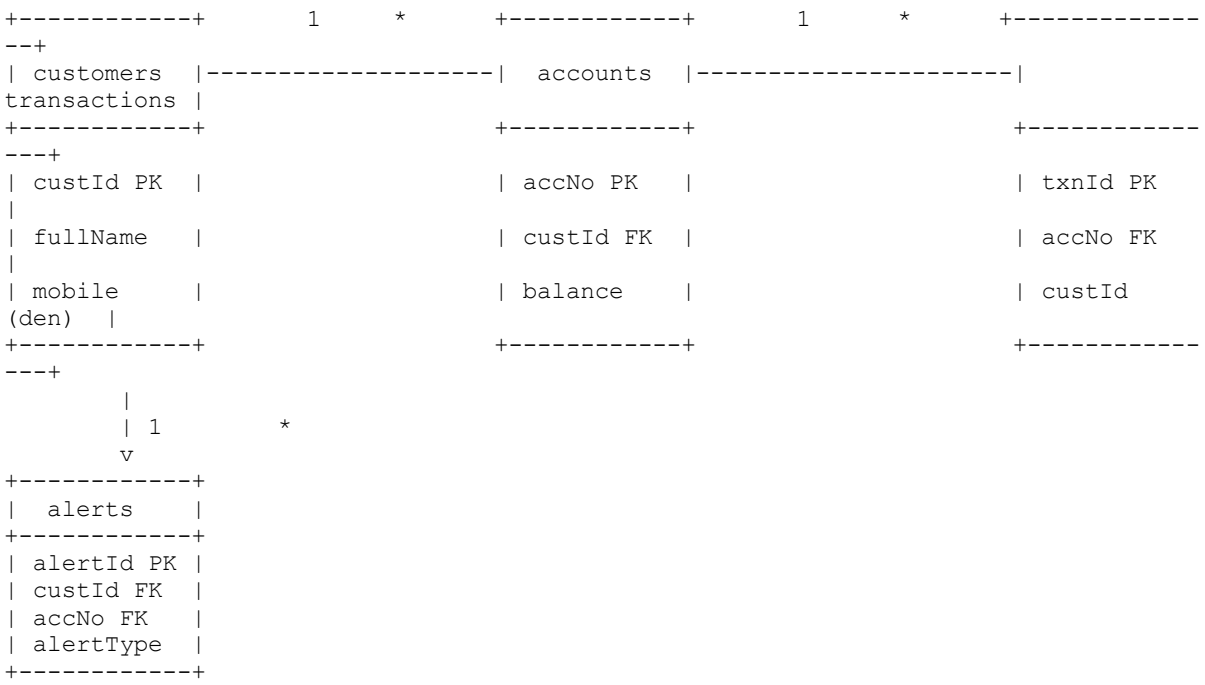
```
CREATE TABLE alerts (
    alert_id        VARCHAR(20) PRIMARY KEY,
    cust_id         VARCHAR(20) REFERENCES customers(cust_id),
    acc_no          VARCHAR(20) REFERENCES accounts(acc_no),
    alert_type      VARCHAR(50),
    severity        VARCHAR(20),
    description     VARCHAR(500),
    threshold_amt   DECIMAL(18,2),
    actual_amt      DECIMAL(18,2),
    txn_count       INT,
    txn_date_only   DATE,
    status          VARCHAR(20),
    created_at      TIMESTAMP,
    updated_at      TIMESTAMP
);
```

B. Relationships & Document-Based ER Diagram

1. Relationships

- **customers → accounts**: 1 to many
- **accounts → transactions**: 1 to many
- **customers/accounts → alerts**: 1 to many

2. ASCII ER (Document-Oriented View)



Learner deliverable: Draw this neatly in an ER tool / PPT as **"Document-Based ER Diagram"**.

C. Indexing Strategy (MongoDB + SQL)

1. customers Indexes

🟢 MongoDB

```
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 });
```

🔵 SQL

```
CREATE UNIQUE INDEX idx_customers_cust_id ON customers(cust_id);
CREATE UNIQUE INDEX idx_customers_mobile  ON customers(mobile);
CREATE UNIQUE INDEX idx_customers_email   ON customers(email);
CREATE INDEX idx_customers_kyc_risk       ON customers(kyc_status,
risk_rating);
```

---

2. accounts Indexes

🟢 MongoDB

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

🔵 SQL

```
CREATE UNIQUE INDEX idx_accounts_acc_no    ON accounts(acc_no);
CREATE INDEX idx_accounts_cust_status      ON accounts(cust_id, status);
```

---

3. transactions Indexes (for Lab2 + Lab3 features)

🟢 MongoDB

```
// Account statement queries
db.transactions.createIndex({ accNo: 1, timestamp: -1 });

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

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

🔵 SQL

```
CREATE INDEX idx_txn_acc_ts      ON transactions(acc_no, ts DESC);
CREATE INDEX idx_txn_cust_ts     ON transactions(cust_id, ts DESC);
CREATE INDEX idx_txn_channel_ts  ON transactions(channel, ts DESC);
```

---

4. alerts Indexes

🟢 MongoDB

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

🔵 SQL

```
CREATE INDEX idx_alerts_cust_created ON alerts(cust_id, created_at DESC);
CREATE INDEX idx_alerts_status_sev   ON alerts(status, severity);
```

D. Capstone Queries – MongoDB + SQL Equivalents

🔷 Feature 1 (Lab1-style): Customer & Accounts View

1. Get Customer by Mobile (search / login)

🟢 MongoDB

```
db.customers.find(
  { mobile: "9876543210" },
  { _id: 0, custId: 1, fullName: 1, email: 1, kycStatus: 1, riskRating: 1 }
);
```

🔵 SQL

```
SELECT cust_id, full_name, email, kyc_status, risk_rating
FROM customers
WHERE mobile = '9876543210';
```

---

2. Fetch All Accounts of a Customer

🟢 MongoDB

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

🔵 SQL

```
SELECT acc_no, type, balance, currency
FROM accounts
WHERE cust_id = 'C1001'
  AND status = 'ACTIVE';
```

---

3. Customer + Accounts in One View (

$lookup ≈ SQL JOIN)

🟢 MongoDB

```
db.customers.aggregate([
  { $match: { custId: "C1001" } },
  {
    $lookup: {
      from: "accounts",
      localField: "custId",
      foreignField: "custId",
      as: "accounts"
    }
  },
  {
    $project: {
      _id: 0,
      custId: 1,
      fullName: 1,
      accounts: { accNo: 1, type: 1, balance: 1, status: 1 }
    }
  }
```

```
]);
```

🔵 SQL

```
SELECT
    c.cust_id,
    c.full_name,
    a.acc_no,
    a.type,
    a.balance,
    a.status
FROM customers c
JOIN accounts a
  ON c.cust_id = a.cust_id
WHERE c.cust_id = 'C1001';
```

🔷 Feature 2 (Lab2-style): Transaction Logging & Reporting

4. Last 5 Transactions of an Account

🟢 MongoDB

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

🔵 SQL

```
SELECT txn_id, type, amount, channel, ts
FROM transactions
WHERE acc_no = 'SAV1001'
  AND status = 'POSTED'
ORDER BY ts DESC
LIMIT 5;
```

---

5. Total Debit/Credit Per Customer (Day2 Aggregations)

🟢 MongoDB

```
db.transactions.aggregate([
  { $match: { status: "POSTED" } },
  {
    $group: {
      _id: "$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
    }
  }
]);
```

🔵 SQL

```
SELECT
    cust_id,
    SUM(CASE WHEN type = 'DEBIT'  THEN amount ELSE 0 END) AS total_debit,
    SUM(CASE WHEN type = 'CREDIT' THEN amount ELSE 0 END) AS total_credit,
    COUNT(*) AS txn_count
FROM transactions
WHERE status = 'POSTED'
GROUP BY cust_id;
```

🔷 Feature 3 (Lab3-style): Fraud Detection Pipeline

6. High-Value DEBIT Transactions (Rule 1)

🟢 MongoDB

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

🔵 SQL

```
SELECT txn_id, cust_id, acc_no, amount, channel, ts
FROM transactions
WHERE type = 'DEBIT'
  AND amount > 100000;
```

---

7. Fraud Rule: Daily Debit > ₹10,00,000 (Core Lab3 Pipeline)

🟢 MongoDB

```
db.transactions.aggregate([
  // 1) Only POSTED DEBITs
  {
```

```
    $match: {
      type: "DEBIT",
      status: "POSTED"
    }
  },
  // 2) Date-only field
  {
    $addFields: {
      txnDateOnly: {
        $dateToString: { format: "%Y-%m-%d", date: "$timestamp" }
      }
    }
  },
  // 3) Group by cust + date
  {
    $group: {
      _id: {
        custId: "$custId",
        txnDateOnly: "$txnDateOnly"
      },
      totalDailyDebit: { $sum: "$amount" },
      txnCount: { $sum: 1 }
    }
  },
  // 4) Apply rule: > 10,00,000
  {
    $match: {
      totalDailyDebit: { $gt: 1000000 }
    }
  },
  // 5) Sort suspicious cases
  {
    $sort: { totalDailyDebit: -1 }
  }
]);
```

🔵 SQL

```
SELECT
    cust_id,
    DATE(ts) AS txn_date_only,
    SUM(amount) AS total_daily_debit,
    COUNT(*)   AS txn_count
FROM transactions
WHERE type = 'DEBIT'
  AND status = 'POSTED'
GROUP BY cust_id, DATE(ts)
HAVING SUM(amount) > 1000000
ORDER BY total_daily_debit DESC;
```

🔷 Feature 4: Customer360 View (Combining All)

8. Complete Customer360 (Profile + Accounts + Last 5 Txns + Alerts)

🟢 MongoDB

```
db.customers.aggregate([
  { $match: { custId: "C1003" } },

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

  // Latest 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"
    }
  },

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

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

  // 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,
      alerts: {
        alertId: 1,
        alertType: 1,
        severity: 1,
        status: 1,
        createdAt: 1
      }
    }
  }
]);
```

🔵 **SQL** (conceptual – typically multiple queries or a big JOIN+subqueries):

```
-- 1) Customer profile
SELECT * FROM customers WHERE cust_id = 'C1003';

-- 2) Accounts
SELECT * FROM accounts WHERE cust_id = 'C1003';

-- 3) Last 5 transactions (across all accounts)
SELECT *
FROM transactions
WHERE cust_id = 'C1003'
  AND status = 'POSTED'
ORDER BY ts DESC
LIMIT 5;

-- 4) Alerts
SELECT *
FROM alerts
WHERE cust_id = 'C1003'
ORDER BY created_at DESC;
```

In SQL world, you often assemble **Customer360** in the **application layer** or using **views**; in MongoDB you can get a **single document** with $lookup pipelines.

E. Capstone Deliverables

1. **Schema Document**
   - Collections / Tables
   - Key fields & data types
   - Indexing strategy + explanation
2. **Document-Based ER Diagram**
   - customers, accounts, transactions, alerts
   - Relationships (1–many)
3. **MongoDB Script**

- create_collections_and_indexes.js
- insert_sample_data.js
- capstone_queries.js (all queries above)

4. **SQL Script**
   - create_tables_indexes.sql
   - insert_sample_data.sql
   - capstone_queries.sql
5. **Short Explanation (2–3 pages)**
   - How Customer360 works
   - How Labs 1, 2, 3 features are integrated:
     - Lab1: Document design, $lookup, basic CRUD
     - Lab2: Transaction logging, summaries
     - Lab3: Fraud detection pipeline feeding alerts
