MoMo_SMS_DATA_EXTRACT Database implementation

1. Overview

This document describes the database design and JSON serialization strategy for the MoMo SMS data processing system. It translates the XML-based transaction feed and business requirements into a normalized MySQL schema designed for data integrity, performance, and future scalability. The core entities are: transactions, users (customers), transaction_categories, and system_logs. A junction table resolves many-to-many relationships where necessary (for example: transactions linked to multiple categories or tags in future extensions). The documentation includes an ERD justification, full table definitions with suggested data types, DDL examples, sample DML (5+ rows per main table), representative CRUD queries and expected results, JSON schema examples showing nested serialization for API responses, and an explicit mapping between SQL tables and JSON structures.

2. ERD

ERD justification :

The XML data feed for MoMo SMS transactions includes sender/receiver details, a transaction identifier, timestamps, amounts, payment/transfer types, and processing metadata. To model this effectively while maintaining normalization and query performance, I designed four core entities: users, transactions, transaction_categories, and system_logs. users stores persistent customer information (MSISDN, name, KYC id, contact metadata) and is referenced by transactions to represent both payer and payee via two foreign keys (sender_id, receiver_id) — this enables efficient joins for user-centric queries and prevents duplication of user details across transactions. transactions is the main fact table: it stores the canonical transaction id (unique business key), timestamps, amount (DECIMAL to preserve precision), status, a JSON raw_payload column to store original XML->JSON blob for replay/audit, and FK to a category.

transaction_categories is a small lookup table (e.g., payment, cash_in, cash_out, bill_payment, airtime) allowing fast grouping and analytics without string scanning. system_logs captures ETL processing events, errors, and transformation metadata with FK to transactions where applicable. Because a transaction may relate to multiple categories/tags in future (e.g., bill_payment + promotion_tag), the ERD includes a resolved many-to-many via transaction_category_map (junction table). Indexes are added on high-cardinality lookup columns such as transactions.transaction_time, transactions.status, and users.msisdn to improve common query patterns. CHECK constraints (MySQL 8+)

enforce basic business rules (positive amount, ISO currency), and column comments document intent.

3. Entities, attributes, keys & relationships

Entities (core)

1. users
    - user_id INT AUTO_INCREMENT PRIMARY KEY
    - msisdn VARCHAR(20) NOT NULL UNIQUE – E.164 phone number
    - first_name VARCHAR(100)
    - last_name VARCHAR(100)
    - email VARCHAR(255)
    - kyc_id VARCHAR(64) – external KYC identifier
    - created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
    - updated_at DATETIME
    - comments: stores registered customers or counterparty info
2. transaction_categories
    - category_id INT AUTO_INCREMENT PRIMARY KEY
    - code VARCHAR(50) NOT NULL UNIQUE – e.g. CASH_IN, PAYMENT
    - description VARCHAR(255)
    - created_at DATETIME DEFAULT CURRENT_TIMESTAMP
3. transactions
    - transaction_id BIGINT AUTO_INCREMENT PRIMARY KEY
    - txn_ref VARCHAR(128) NOT NULL UNIQUE – business/third-party transaction id
    - sender_id INT NOT NULL – FK -> users(user_id)
    - receiver_id INT – FK -> users(user_id) (nullable for merchant-less flows)
    - amount DECIMAL(18,2) NOT NULL – monetary amount

- currency CHAR(3) NOT NULL DEFAULT ‘RWF’ – ISO 4217
- status ENUM(‘PENDING’,‘SUCCESS’,‘FAILED’,‘REVERSED’) NOT NULL DEFAULT ‘PENDING’
- transaction_time DATETIME NOT NULL
- category_id INT NOT NULL – FK -> transaction_categories(category_id)
- raw_payload JSON – stores original parsed XML or raw payload for audit
- processed_at DATETIME – when ETL processed it
- created_at DATETIME DEFAULT CURRENT_TIMESTAMP
- CHECK (amount > 0)

4. transaction_category_map (junction table for many-to-many)
   - txn_cat_map_id INT AUTO_INCREMENT PRIMARY KEY
   - transaction_id BIGINT NOT NULL – FK -> transactions(transaction_id)
   - category_id INT NOT NULL – FK -> transaction_categories(category_id)
   - UNIQUE(transaction_id, category_id)
5. system_logs
   - log_id BIGINT AUTO_INCREMENT PRIMARY KEY
   - event_time DATETIME DEFAULT CURRENT_TIMESTAMP
   - level ENUM(‘INFO’,‘WARN’,‘ERROR’,‘DEBUG’) NOT NULL DEFAULT ‘INFO’
   - source VARCHAR(100) – e.g. ETL, API, Validator
   - message TEXT
   - transaction_id BIGINT NULL – optional FK -> transactions(transaction_id)
   - metadata JSON – arbitrary metadata

Relationships & cardinality (textual) - users 1 — M transactions as sender (sender_id) (1:M) - users 1 — M transactions as receiver (receiver_id) (1:M) - transactions M — N transaction_categories resolved by transaction_category_map (M:N) - transactions 1 — M system_logs (processing logs may reference transactions)

4. SQL Implementation (database_setup.sql)

Note: The following DDL is a documentation excerpt. Use it as the basis for database_setup.sql.

```
DROP TABLE IF EXISTS System_Logs;

DROP TABLE IF EXISTS Transactions;

DROP TABLE IF EXISTS Transaction_categories;

DROP TABLE IF EXISTS Customers;


 CREATE TABLE Customers (

        customer_id INT PRIMARY KEY AUTO_INCREMENT,

        fullName VARCHAR(50) NOT NULL,

        phoneNumber VARCHAR(12) NOT NULL UNIQUE

);

-- Index for faster lookups by phoneNumber

CREATE INDEX idx_customers_phone ON Customers(phoneNumber);

 CREATE TABLE Transaction_categories (

        category_id INT PRIMARY KEY AUTO_INCREMENT,

        categoryName VARCHAR(40) NOT NULL,

        description VARCHAR(30)

);


-- Index for faster lookups by categoryName

CREATE INDEX idx_category_name ON Transaction_categories(categoryName);

 CREATE TABLE Transactions (

        transaction_id INT PRIMARY KEY AUTO_INCREMENT,

        partyA VARCHAR(12) NOT NULL,
```

```
    partyB VARCHAR(12) NOT NULL,
    transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    amount NUMERIC(12,2) NOT NULL,
    category_id INT,
    CONSTRAINT fk_partyA FOREIGN KEY (partyA)
    REFERENCES Customers(phoneNumber)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
    CONSTRAINT fk_partyB FOREIGN KEY (partyB)
    REFERENCES Customers(phoneNumber)
    ON DELETE CASCADE
  ON UPDATE CASCADE,
    CONSTRAINT fk_category FOREIGN KEY (category_id)
    REFERENCES Transaction_categories(category_id)
    ON DELETE SET NULL
    ON UPDATE CASCADE
);

-- Indexes for common queries
CREATE INDEX idx_transactions_partyA ON Transactions(partyA);
CREATE INDEX idx_transactions_partyB ON Transactions(partyB);
CREATE INDEX idx_transactions_category ON Transactions(category_id);

 CREATE TABLE System_Logs (
    log_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT NOT NULL,
    category VARCHAR(40) NOT NULL,
```

```
    transaction_id INT,
    transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_log_customer FOREIGN KEY (customer_id)
    REFERENCES Customers(customer_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
    CONSTRAINT fk_log_transaction FOREIGN KEY (transaction_id)
    REFERENCES Transactions(transaction_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

-- Index for faster log retrieval
CREATE INDEX idx_logs_customer ON System_Logs(customer_id);
CREATE INDEX idx_logs_transaction ON System_Logs(transaction_id);
```

Sample DML (insert at least 5 rows per main table)

```
-- Customers
INSERT INTO Customers (fullName, phoneNumber) VALUES
('Alice Johnson', '0700000001'),
('Bob Smith', '0700000002'),
('Charlie Brown', '0700000003'),
('Diana Prince', '0700000004'),
('Ethan Hunt', '0700000005');
 -- Transaction Categories
INSERT INTO Transaction_categories (categoryName, description) VALUES
('Deposit', 'Money deposited'),
```

```
('Withdrawal', 'Money withdrawn'),

('Transfer', 'Money transferred'),

('Payment', 'Payment made'),

('Refund', 'Money refunded');

 -- Transactions

INSERT INTO Transactions (partyA, partyB, amount, category_id) VALUES

('0700000001', '0700000002', 100.00, 1),

('0700000003', '0700000004', 250.50, 3),

('0700000002', '0700000005', 75.25, 2),

('0700000004', '0700000001', 300.00, 4),

('0700000005', '0700000003', 150.00, 5);

 -- system_logs (5)

INSERT INTO System_Logs (customer_id, category, transaction_id) VALUES

(1, 'Deposit', 1),

(2, 'Transfer', 1),

(3, 'Transfer', 2),

(4, 'Transfer', 2),

(5, 'Withdrawal', 3);
```

5. Representative CRUD queries & expected results

Create (already shown via INSERT above)

Read (example): Get full transaction + user info

```
SELECT t.txn_ref, t.amount, t.currency, t.status, t.transaction_time,
   s.msisdn AS sender_msisdn, s.first_name AS sender_first, s.last_name AS sender_last,
   r.msisdn AS receiver_msisdn, r.first_name AS receiver_first, r.last_name AS receiver_last,
   c.code AS category_code
 FROM transactions t
 JOIN users s ON t.sender_id = s.user_id
 LEFT JOIN users r ON t.receiver_id = r.user_id
```

```
JOIN transaction_categories c ON t.category_id = c.category_id
WHERE t.txn_ref = 'REF-20250922-0001';
```

Expected single-row result (illustrative):

- txn_ref: REF-20250922-0001
- amount: 15000.00
- currency: RWF
- status: SUCCESS
- transaction_time: 2025-09-22 10:05:00
- sender_msisdn: +250788000001
- receiver_msisdn: +250788000002
- category_code: P2P

Update (example): set transaction status to SUCCESS and processed_at)

```
UPDATE transactions
SET status = 'SUCCESS', processed_at = NOW()
WHERE txn_ref = 'REF-20250922-0002';
```

Delete (example): logically delete by archiving raw_payload then deleting)

```
-- Archive pattern (application-level recommended)
DELETE FROM transactions WHERE transaction_id = 999; -- use with caution
```

6. JSON Data Modeling & mapping

Mapping principles: - users → serialized as user object with user_id, msisdn, first_name, last_name, email. - transaction_categories → category object with category_id, code, description. - transactions → top-level transaction object. sender and receiver are embedded user objects (not just IDs) for convenience in API responses. raw_payload is included as raw_payload (JSON) for audit/debug clients. - system_logs → separate logs array or log endpoints; when included inline, only recent relevant logs are serialized.

Example JSON for a single complete transaction (complex object):

```
{
  "transaction": {
        "transaction_id": 1,
        "txn_ref": "REF-20250922-0001",
        "amount": 15000.00,
        "currency": "RWF",
```

```
        "status": "SUCCESS",
        "transaction_time": "2025-09-22T10:05:00Z",
        "category": {
        "category_id": 3,
        "code": "P2P",
        "description": "Peer-to-peer transfer"
        },
        "sender": {
        "user_id": 1,
        "msisdn": "+250788000001",
        "first_name": "Innocent",
        "last_name": "Muvunyi",
        "email": "innocent@example.com"
        },
        "receiver": {
        "user_id": 2,
        "msisdn": "+250788000002",
        "first_name": "Alice",
        "last_name": "Uwase",
        "email": "alice@example.com"
        },
        "raw_payload": {
        "source": "sms",
        "raw": "<xml>...1</xml>"
        },
        "processed_at": "2025-09-22T10:05:02Z"
  }
}
```


JSON schema snippets (informal):

- User:
    - user_id (integer)
    - msisdn (string)
    - first_name (string)
    - last_name (string)
    - email (string|null)
- Transaction:
    - transaction_id (integer)
    - txn_ref (string)

- amount (number)
- currency (string)
- status (string)
- transaction_time (string, ISO8601)
- category (object)
- sender (User object)
- receiver (User object|null)
- raw_payload (object|null)