MoMo SMS Data Processing System Database Design Document

Team: EWDGroup-13
Course: Enterprise Web Development
Institution: African Leadership University Rwanda
Date: January 26, 2026

Team Members:

1. Igor Ntwali
2. Erioluwa Gideon Olowoyo
3. Cyuzuzo Germain
4. Okechukwu Wisdom Ikechukwu
5. Kellen Ashley Mutoni

Introduction

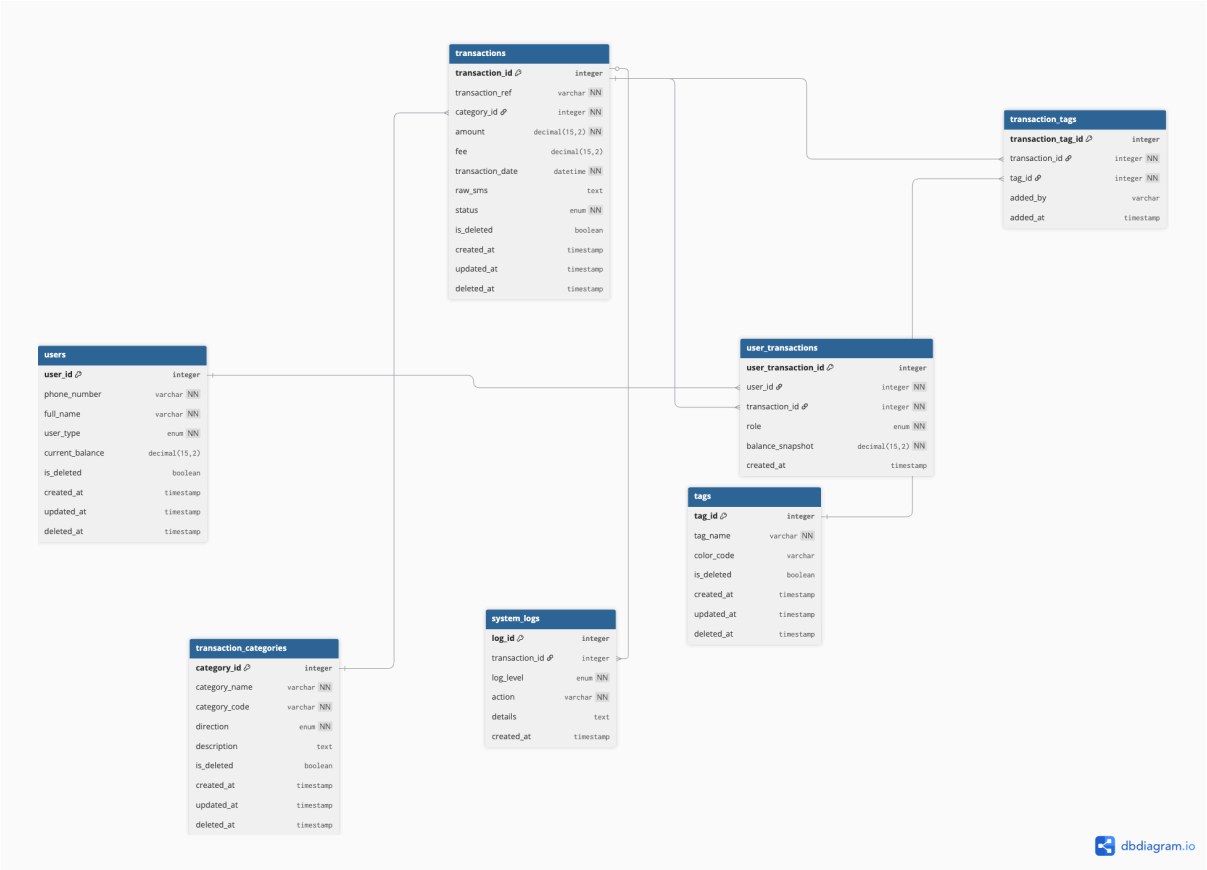
The MoMo SMS Data Processing System is designed to efficiently store and manage mobile money transaction data extracted from SMS messages in XML format. As mobile money services continue to grow, analyzing transaction patterns has become essential for understanding user behavior, improving financial services, and generating actionable insights.

This database serves as the foundation for organizing and analyzing transaction data, enabling reliable reporting and supporting future scalability. It is designed to handle nine types of transactions, including money transfers, deposits, withdrawals, merchant payments, and service purchases, while maintaining data integrity and performance.

The dataset powering this system consists of 1,694 SMS records from the MTN Mobile Money (MoMo) service in Rwanda. Each record contains detailed information such as transaction amount, sender and receiver details, transaction date, and a unique reference number. By translating this raw SMS data into a structured database, the system ensures that transaction information is accessible, consistent, and ready for analysis.

Database Entities and Relationships

To provide a clear overview of how the data entities relate to each other and to illustrate the structure of the database, an **Entity Relationship Diagram (ERD)** has been developed and is presented in the following section.



Looking at the database design presented in the **Entity Relationship Diagram (ERD)** above, our system is structured around six core entities, each serving a specific role in storing and organizing MoMo SMS transaction data:

1. **Users** – Stores information about individuals who send or receive money.
2. **Transactions** – The primary table that records all transaction details.
3. **Transaction_Categories** – Represents the nine types of transactions identified, such as transfers, deposits, and payments.
4. **User_Transactions** – A junction table that captures user participation in transactions, indicating each user's role (sender or receiver) and storing balance snapshots after each transaction.
5. **System_Logs** – Captures logs generated during the processing of the XML data, supporting audit and debugging purposes.
6. **Tags** – Contains labels that can be applied to transactions for categorization or tracking (e.g., "verified" or "high-value").
7. **Transaction_Tags** – A junction table that links transactions to tags, resolving the many-to-many relationship between these entities.

Database Table Relationships

The table is as below:

Table	Column	Referrences	Relationship Type	Notes
transactions	category_id	transaction_categories.category_id	Many-to-One	Each transaction belongs to one category
transaction_tags	transaction_id	transactions.transaction_id	Many-to-One	Links transaction to tags
transaction_tags	tag_id	tags.tag_id	Many-to-One	Links tag to transactions
user_transactions	user_id	users.user_id	Many-to-One	Maps users to transactions
user_transactions	transaction_id	transactions.transaction_id	Many-to-One	Maps transactions to users
system_logs	transaction_id	transactions.transaction_id	Many-to-One	Optional log related to a transaction

The database entities are connected through the following relationships:

The **Transaction_Tags** table is a junction table that resolves the many-to-many relationship between transactions and tags. This design enables:

- A single transaction to have multiple tags (for example, both “verified” and “high-value”).
- A single tag to be associated with multiple transactions (for example, the “verified” tag can be applied to many transactions).

The junction table uses a **composite primary key** consisting of transaction_id and tag_id to ensure the uniqueness of each transaction-tag pair.

Design Rationale and Justification

Our database design for the MoMo SMS Data Processing System was guided by principles of **organization, data integrity, and query efficiency**, ensuring that transaction data is stored in a structured and easily accessible manner. The following design decisions highlight the rationale behind the structure of our database:

Users Table

A single **Users** table stores all individuals involved in transactions, including senders, receivers, and merchants. This approach prevents duplication of phone numbers and centralizes user information. The user_type field distinguishes between CUSTOMERS, MERCHANTS, and AGENTS, allowing the system to identify the role of each user. Transactions reference the Users table through foreign keys, ensuring consistency and eliminating redundancy.

Transaction Categories

Analysis of the SMS data revealed nine distinct types of transactions. To manage this efficiently, a separate **Transaction_Categories** table was created instead of storing category names directly in each transaction record. This design reduces errors from inconsistent naming, simplifies the addition of new categories, and provides space to store extra information, such as the detection patterns used to identify each transaction type.

Many-to-Many Relationships with Tags

To support flexible labeling of transactions for analysis, a **Transaction_Tags** junction table was implemented. This allows multiple tags to be associated with a single transaction without modifying the database schema. For example, a large incoming transfer can simultaneously be tagged as “verified” and “high-value.” This design aligns with database normalization principles and enhances the system’s analytical capabilities.

System Logs

The **System_Logs** table captures events that occur during the processing of SMS data. Logs can be linked to specific transactions via foreign keys or recorded as general system entries without a transaction reference. This ensures traceability, aids in debugging, and allows monitoring of data processing operations.

Data Types In Our Database

Appropriate data types were selected to ensure accuracy and compatibility:

1. **DECIMAL(15,2)** for monetary amounts prevents rounding errors associated with floating-point numbers.
2. **VARCHAR(15)** for phone numbers accommodates Rwanda's phone format (250XXXXXXXXX) while preserving leading zeros.
3. **TEXT** for storing original SMS messages ensures that parsed data can be verified against the source.

Indexes

Indexes were added on frequently queried columns, such as transaction_date and phone_number. These indexes improve query performance, particularly for date-range filtering and user-specific transaction lookups.

Overall, these design choices ensure that the database is **normalized, efficient, and scalable**, supporting accurate analysis and reporting of mobile money transactions while maintaining flexibility for future enhancements.

Data Dictionary

The following section provides a detailed description of all database tables, their columns, data types, constraints, and purpose within the MoMo SMS Data Processing System. This dictionary ensures clarity for developers and stakeholders and supports accurate data handling.

Table: users

Description: Stores information about individuals who send or receive money through MoMo.

Column	Data Type	Constraints	Description
user_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each user
phone_number	VARCHAR(15)	UNIQUE, NOT NULL	Phone number in Rwanda format (250XXXXXXXXX)
full_name	VARCHAR(100)	NOT NULL	Person's name as it

			appears in SMS
masked_phone	VARCHAR(15)	NULL	Masked phone format from SMS (*********XXX)
user_type	ENUM	NOT NULL, DEFAULT 'CUSTOMER'	Type of user: CUSTOMER, MERCHANT, or AGENT
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Record creation timestamp
updated_at	DATETIME	ON UPDATE CURRENT_TIMESTAMP	Last modification timestamp

Table: transaction_categories

Description: Stores the nine types of MoMo transactions identified from analyzing SMS data.

Column	Data Type	Constraints	Description
category_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each category
category_name	VARCHAR(50)	UNIQUE, NOT NULL	Human-readable name (e.g; “Incoming Transfer”)
category_code	VARCHAR(20)	UNIQUE, NOT NULL	Short code for the category (e.g., "INCOMING_TRANSFER")

pattern_prefix	VARCHAR(100)	NULL	Text pattern used to detect this type
direction	ENUM	NOT NULL	CREDIT (money in) or DEBIT (money out)
description	TEXT	NULL	
is_active	BOOLEAN	DEFAULT TRUE	
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	

Table: transactions

Description: Stores all MoMo transaction records extracted from SMS messages.

Column	Data Type	Constraints	Description
transaction_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each transaction
transaction_ref	VARCHAR(50)	UNIQUE, NOT NULL	Transaction ID from SMS
external_ref	VARCHAR(50)	NULL	External reference for third-party transactions
sender_id	INT	FOREIGN KEY	References users.user_id (sender)
receiver_id	INT	FOREIGN KEY	References users.user_id (receiver)
category_id	INT	FOREIGN KEY, NOT NULL	References transaction_categori

			es.category_id
amount	DECIMAL(15,2)	NOT NULL, CHECK > 0	Transaction amount in RWF
fee	DECIMAL(15,2)	DEFAULT 0, CHECK >= 0	Fee charged for the transaction
balance_after	DECIMAL(15,2)	NULL	Account balance after transaction
transaction_date	DATETIME	NOT NULL	Timestamp of transaction
raw_sms_body	TEXT	NOT NULL	Original SMS message text
sms_address	VARCHAR(20)	NULL	SMS sender address (usually "M-Money")
status	ENUM	DEFAULT 'COMPLETED'	Transaction status: COMPLETED, PENDING, FAILED
token	VARCHAR(100)	NULL	Token for utility payments
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Record creation timestamp

Table: tags

Description: Stores labels that can be applied to transactions for flexible categorization.

Column	Data Type	Constraints	Description
tag_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each tag
tag_name	VARCHAR(50)	UNIQUE, NOT NULL	Tag name (e.g., "verified", "high-value")
color_code	VARCHAR(7)	NULL	Hex color for display (e.g., "#28A745")
description	TEXT	NULL	Explanation of the tag
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Record creation timestamp

Table: transaction_tags (Junction Table)

Description: Resolves the many-to-many relationship between transactions and tags.

Column	Data Type	Constraints	Description
transaction_id	INT	PRIMARY KEY, FOREIGN KEY	References transactions.transaction_id
tag_id	INT	PRIMARY KEY, FOREIGN KEY	References tags.tag_id
assigned_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Timestamp when the tag was applied

Table:system_logs

Description: Stores logs generated during ETL processing of SMS data.

Column	Data Type	Constraints	Description
log_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each log entry
transaction_id	INT	FOREIGN KEY, NULL	Related transaction
log_level	ENUM	DEFAULT “INFO”	Severity level: DEBUG, INFO, WARNING, ERROR
action	VARCHAR (50)	NOT NULL	Action performed
details	TEXT	NULL	Additional information about the action
source_file	VARCHAR(255)	NULL	Name of XML file processed
line_number	INT	NULL	Line number in source file
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Log creation timestamp

Table: user_transactions (Junction Table)

Description: Captures the participation of users in transactions, indicating whether a user acted as a sender or receiver and storing the user’s balance snapshot after the transaction.

Column	Data Type	Constraints	Description
user_transaction_id	INT	PRIMARY KEY	Unique identifier for each user–transaction mapping
user_id	INT	FOREIGN KEY, NOT NULL	References users.user_id
transaction_id	INT	FOREIGN KEY, NOT NULL	References transactions.transaction_id
role	ENUM	NOT NULL	Role of the user in the transaction: sender or receiver
balance_snapshot	DECIMAL(15,2)	NOT NULL	User's account balance immediately after the transaction
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	When the mapping record was created

Sample Queries With Results

To demonstrate the functionality of our MoMo SMS Data Processing System database, the following SQL queries were executed.

Query 1: View All Transactions

This query retrieves the most recent transactions along with their associated category name. It helps in quickly viewing transaction details and verifying proper linkage between transactions and transaction_categories.

```
============================================================
QUERY1_VIEW_ALL_TRANSACTIONS
============================================================
 transaction_id | transaction_ref |          amount |             fee | transaction_date |   category_name
------------------------------------------------------------------------------------------------------------
            714 |     37832903831 |         24900.0 |             0.0 | 2025-01-16 00:1 | Payment to Code
            713 |     51350491173 |          1500.0 |             0.0 | 2025-01-15 20:3 | Payment to Code
            712 |     26811810649 |         27000.0 |             0.0 | 2025-01-15 20:2 | Payment to Code
            711 |     55109401801 |         14500.0 |             0.0 | 2025-01-14 21:2 | Payment to Code
            710 |     30801532894 |          1000.0 |             0.0 | 2025-01-14 20:0 | Payment to Code
```

Query 2: Count Transactions by Category

This query provides a summary of transactions grouped by their category, including the direction (inbound or outbound), number of transactions, and the total transaction amount.

```
============================================================
QUERY2_COUNT_BY_CATEGORY
============================================================
  category_name |       direction | transaction_count |    total_amount
------------------------------------------------------------------------
Payment to Code |        outbound |               660 |       4811892.0
Incoming Transf |         inbound |                38 |         15753.0
Third Party Deb |        outbound |                 9 |          4000.0
Bundle Purchase |        outbound |                 6 |          2400.0
Airtime Purchas |        outbound |                 1 |           200.0
```

Query 3: Count Records in Each Table

This query verifies that data has been successfully loaded into all major tables, ensuring completeness of the database.

```
============================================================
QUERY3_TABLE_COUNTS
============================================================
     table_name |         records
--------------------------------
          users |               1
   transactions |             714
transaction_cat |               9
           tags |               5
    system_logs |               2
```

Security Rules

To ensure **data integrity**, **accuracy**, and **security**, the database enforces the following rules:

```
● $ mysql -u root < database/database_setup_sql.sql 2>&1

⊗ $ mysql -u root -e "
  ERROR 3819 (HY000) at line 4: Check constraint 'chk_phone_format' is violated.

⊗ $ mysql -u root -e "
  ERROR 3819 (HY000) at line 4: Check constraint 'chk_amount_positive' is violated.

⊗ $ mysql -u root -e "
  ERROR 1062 (23000) at line 6: Duplicate entry 'UNIQUE123' for key 'transactions.transaction_ref'
```

1. Phone Number Format Validation

- **Description:** Ensures all phone numbers follow Rwanda's format: `250` + 9 digits.
- **SQL Implementation:**

```
CONSTRAINT chk_phone_format CHECK (phone_number REGEXP
'^250[0-9]{9}$')
```

- **Purpose:** Prevents invalid phone numbers from being stored.
- **Example:**

```
INSERT INTO users (phone_number, full_name)
VALUES ('123456789', 'Test User');
```

Result: **Error:** check constraint violated.

---

2. Positive Transaction Amount

- **Description:** Ensures that transaction amounts are always greater than zero.
- **SQL Implementation:**

```
CONSTRAINT chk_amount_positive CHECK (amount > 0)
```

- **Purpose:** Prevents zero or negative transaction entries.
- **Example:**

```
INSERT INTO transactions (transaction_ref, category_id,
amount, transaction_date)
VALUES ('TEST001', 1, 0, NOW());
```

Result: **Error:** check constraint violated.

---

3. Unique Transaction References

- **Description:** Ensures each transaction has a unique reference number.
- **SQL Implementation:**

```
transaction_ref VARCHAR(50) NOT NULL UNIQUE
```

- **Purpose:** Prevents duplicate transactions.
- **Example:**

```
INSERT INTO transactions (transaction_ref, category_id,
amount, transaction_date)
VALUES ('UNIQUE123', 1, 1000, NOW());
INSERT INTO transactions (transaction_ref, category_id,
amount, transaction_date)
VALUES ('UNIQUE123', 1, 2000, NOW());
```

Result: **Error:** duplicate entry.

These **validation rules** enhance data reliability by enforcing correct phone number formats, positive transaction amounts, and unique transaction references. Together, they support **accurate reporting**, **secure processing**, and **future scalability** of the MoMo SMS Data Processing System.