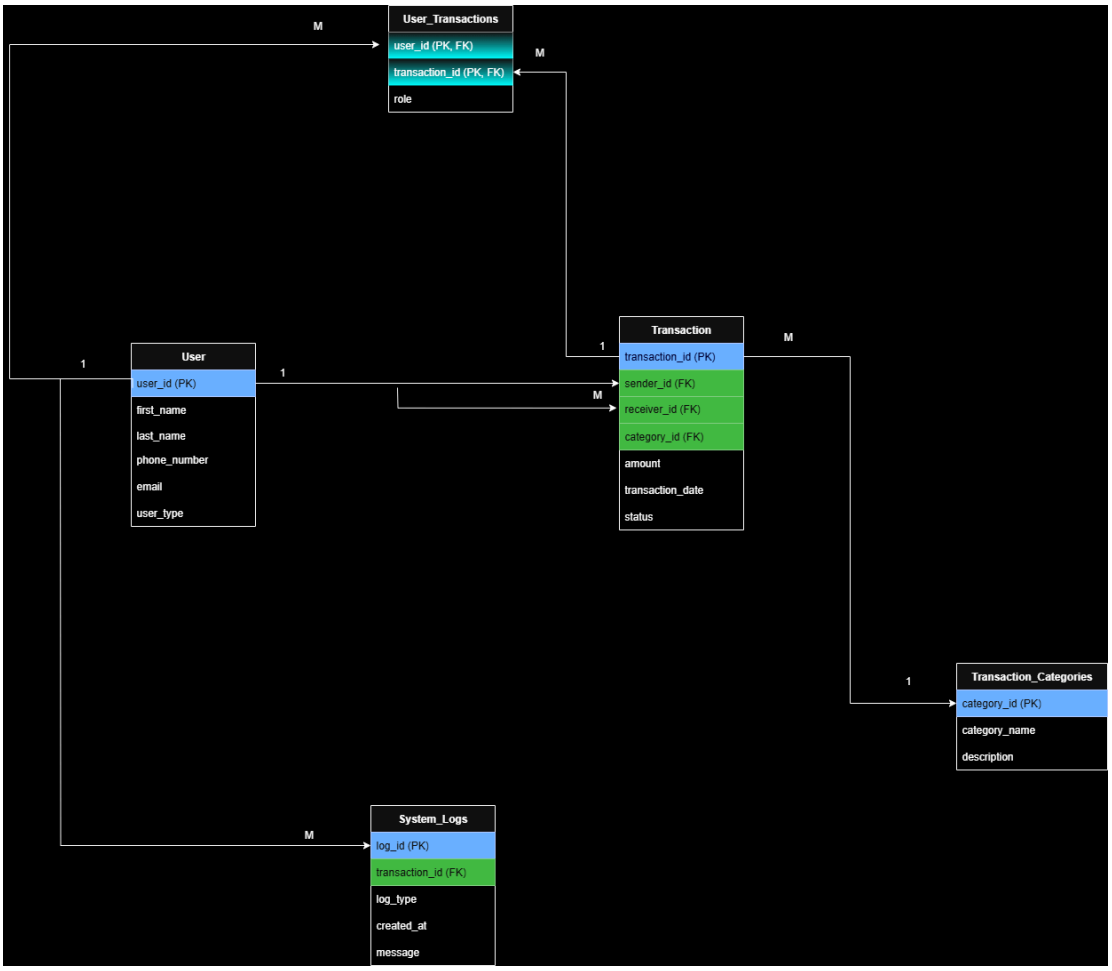
MOMO SMS Data Processing System

The MoMo SMS Data Processing System database is designed to store, manage, and analyze mobile money transactions efficiently. It captures essential details such as users, transactions, categories, and system logs while ensuring data integrity, scalability, and ease of analysis. This document outlines the database design, including the ERD, rationale, data dictionary, and sample queries, serving as a foundation for reliable transaction processing.

Entity Relationship Diagram (ERD)



Design Rationale and Justification

The database schema was designed to support the core functionality of a MoMo SMS data processing system, with a focus on normalization, data integrity, scalability, and analytical flexibility. At the heart of the schema is the ***`Transactions`*** table, which serves as the central entity because it captures the primary business activity—mobile money transfers. Every other table connects to ***`Transactions`***, either directly or through a junction, making it the logical hub for data relationships.

To avoid redundancy and ensure normalization, supporting entities were separated into distinct tables. The ***Users*** table stores sender and receiver information, allowing user details to be reused across multiple transactions without duplication. ***Transaction_Categories*** classifies each transaction by type (e.g., airtime, bill payment), enabling consistent categorization and easy aggregation. ***System_Logs*** tracks backend processing events such as retries or errors, providing transparency and auditability without cluttering the transaction records.

To resolve the many-to-many relationship between users and transactions—where a user can participate in multiple transactions and a transaction can involve multiple users—a junction table called ***User_Transactions*** was introduced. This table links users to transactions and includes a *role* attribute to distinguish between senders, receivers, and other participants.

Foreign keys are used throughout the schema to enforce referential integrity. For example, ***user_id*** in ***Transactions*** and ***System_Logs*** references the ***Users*** table, ensuring that all transactional and log data is tied to valid users.

This design supports scalability by allowing new transaction types, user roles, or log events to be added without restructuring the database. It also enables powerful analysis—such as aggregating transactions by category, user, or date—making it suitable for both operational reporting and strategic insights.

Data Dictionary

Table: Users

Field name	Data Type	Constraints	Description
user_id	INT	PK, AUTO_INCREMENT	Unique identifier for each user
first_name	VARCHAR(50)	NOT NULL	First name of the user
last_name	VARCHAR(50)	NOT NULL	Last name of the user
phone_number	VARCHAR(15)	UNIQUE,NOT NULL	Mobile number used for MOMO transactions
email	VARCHAR	NULL	Optional email address
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Date and time the user was added to the system

Table: Transaction

Field name	Data Type	Constraints	Description
transaction_id	INT	PK,AUTO_INCREMENT	Unique identifier for each transaction
amount	DECIMAL(10,2)	CHECK(amount > 0), NOT NULL	Amount transferred
transaction_date	DATETIME	NOT NULL	Date and time of the transaction
sender_id	INT	FK → Users(user_id), NOT NULL	Sender's user ID
receiver_id	INT	FK → Users(user_id), NOT NULL	Receiver's user ID
category_id	INT	FK → Transaction_Categories(category_id), NOT NULL	Category of transaction
status	VARCHAR(20)	DEFAULT 'Completed'	Status (Completed, Pending, Failed)

Table: User_Transactions

Fieldname	Data type	Constraints	Description
user_transaction_id	INT	PK, AUTO_INCREMENT	Unique identifier
user_id	INT	FK → Users(user_id), NOT NULL	Linked user
transaction_id	INT	FK → Transactions(transaction_id), NOT NULL	Linked transaction
role	ENUM('Sender','Receiver')	NOT NULL	Role of the user in the transaction

Table: Transaction_Categories

Field name	Data type	Constraints	Description
category_id	INT	PK, AUTO_INCREMENT	Unique identifier for each transaction category
category_name	VARCHAR(50)	NOT NULL, UNIQUE	Name of the category (e.g., “Payment”, “Airtime”, “Transfer”)
description	TEXT	NULL	Optional detailed explanation

Table: System_Logs

Field name	Data type	Constraints	Description
log_id	INT	PK, AUTO_INCREMENT	Unique log entry identifier
transaction_id	INT	FK → Transactions(transaction_id), NULL	Related transaction
log_type	VARCHAR(20)	NOT NULL	Log type (INFO, WARNING, ERROR)
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Log timestamp
message	TEXT	NOT NULL	Detailed log message

Sample Queries with Screenshots

Below are screenshots of some examples of queries using CRUD operation;

```
mysql> SELECT * FROM Transaction WHERE amount > 100;
+----------------+-----------+-------------+-------------+----------+---------------------+-
----------+
| transaction_id | sender_id | receiver_id | category_id | amount   | transaction_date    |
status    |
+----------------+-----------+-------------+-------------+----------+---------------------+-
----------+
|           1001 |         1 |           2 |           3 |  5000.00 | 2025-09-17 10:00:00 |
completed |
|           1002 |         2 |           3 |           2 |  2000.00 | 2025-09-17 11:00:00 |
pending   |
|           1003 |         3 |           4 |           1 | 10000.00 | 2025-09-17 12:00:00 |
failed    |
```

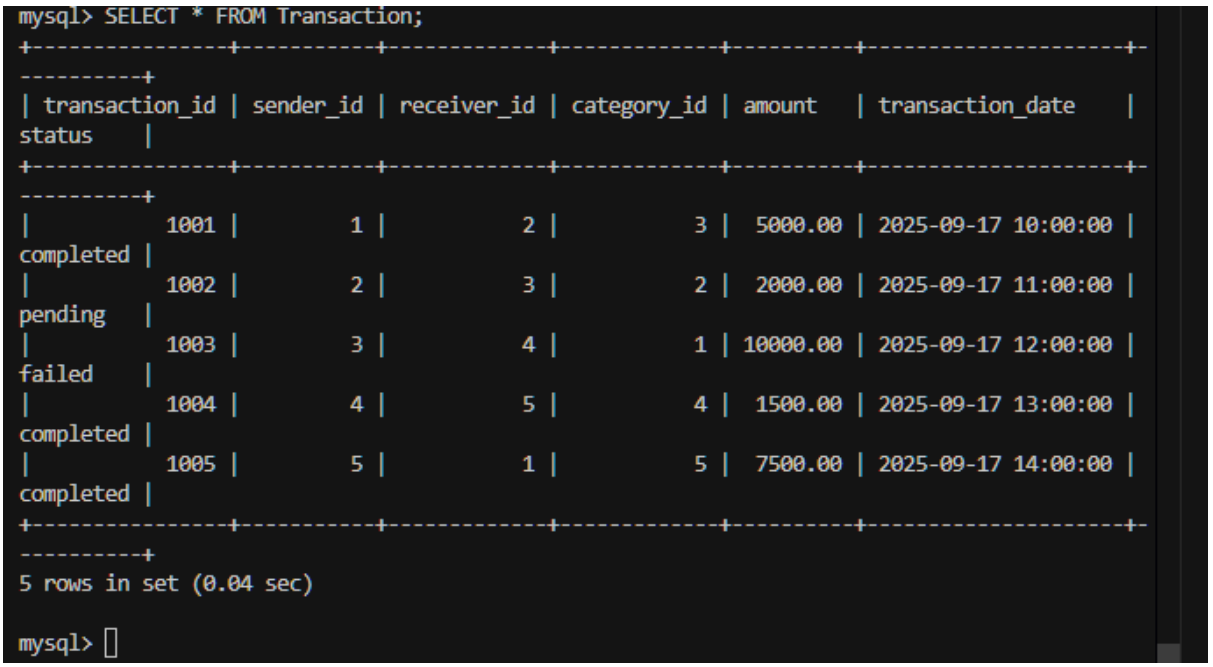
```
mysql> DELETE FROM System_Logs WHERE log_id = 5;
Query OK, 0 rows affected (0.47 sec)

mysql>
```

```
mysql> UPDATE User SET phone_number = '07999' WHERE user_id = 1;
Query OK, 1 row affected (0.28 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

```
mysql> SELECT * FROM Transaction WHERE status = 'Completed';
+----------------+-----------+-------------+-------------+---------+---------------------+--
---------+
| transaction_id | sender_id | receiver_id | category_id | amount  | transaction_date    | s
tatus    |
+----------------+-----------+-------------+-------------+---------+---------------------+--
---------+
|           1001 |         1 |           2 |           3 | 5000.00 | 2025-09-17 10:00:00 | c
ompleted |
|           1004 |         4 |           5 |           4 | 1500.00 | 2025-09-17 13:00:00 | c
ompleted |
|           1005 |         5 |           1 |           5 | 7500.00 | 2025-09-17 14:00:00 | c
ompleted |
+----------------+-----------+-------------+-------------+---------+---------------------+--
---------+
3 rows in set (0.04 sec)
```



```
mysql> SELECT * FROM Transaction;
+----------------+-----------+-------------+-------------+---------+---------------------+-
----------+
| transaction_id | sender_id | receiver_id | category_id | amount  | transaction_date    |
status    |
+----------------+-----------+-------------+-------------+---------+---------------------+-
----------+
|           1001 |         1 |           2 |           3 |  5000.00 | 2025-09-17 10:00:00 |
completed |
|           1002 |         2 |           3 |           2 |  2000.00 | 2025-09-17 11:00:00 |
pending   |
|           1003 |         3 |           4 |           1 | 10000.00 | 2025-09-17 12:00:00 |
failed    |
|           1004 |         4 |           5 |           4 |  1500.00 | 2025-09-17 13:00:00 |
completed |
|           1005 |         5 |           1 |           5 |  7500.00 | 2025-09-17 14:00:00 |
completed |
+----------------+-----------+-------------+-------------+---------+---------------------+-
----------+
5 rows in set (0.04 sec)

mysql>
```

Security and Accuracy Rules

To ensure that the MoMo SMS Data Processing System database remains secure, accurate, and reliable, the following rules and constraints were applied:

1. **Primary and Foreign Keys**
    - Each table includes a primary key to uniquely identify records.
    - Foreign key constraints ensure that relationships between tables remain valid (e.g., a transaction must link to an existing user).
2. **CHECK Constraints**
    - Used in the *`Transaction`* table to prevent invalid data.
    - Example: `CHECK (amount > 0)` ensures that only positive transaction amounts are recorded.
3. **NOT NULL Constraints**
    - Critical fields like *`sender_id`*, *`receiver_id`*, and *`transaction_date`* cannot be empty, ensuring that essential data is always captured.

4. **Unique Constraints**
    - Fields such as *phone_number* in the *Users* table are unique to avoid duplicate user records.
5. **Role-based Accuracy**
    - The *User_Transactions* junction table includes a *role* field restricted to `Sender` or `Receiver` using an ENUM. This guarantees that a user's role in a transaction is always clearly defined.
6. **Default Values**
    - Fields such as `status` in `Transactions` default to `'Completed'`, and `created_at` timestamps default to the current date/time, reducing human error and ensuring consistent data logging.
7. **Referential Integrity**
    - All foreign keys are enforced with `ON DELETE CASCADE` or `ON DELETE RESTRICT` rules (depending on context) to maintain consistency when records are removed.
8. **Auditability**
    - The *System_Logs* table records system events with timestamps and messages, providing traceability and enhancing database security.

Together, these measures protect against invalid entries, prevent duplication, and ensure that transaction data is both trustworthy and secure.