

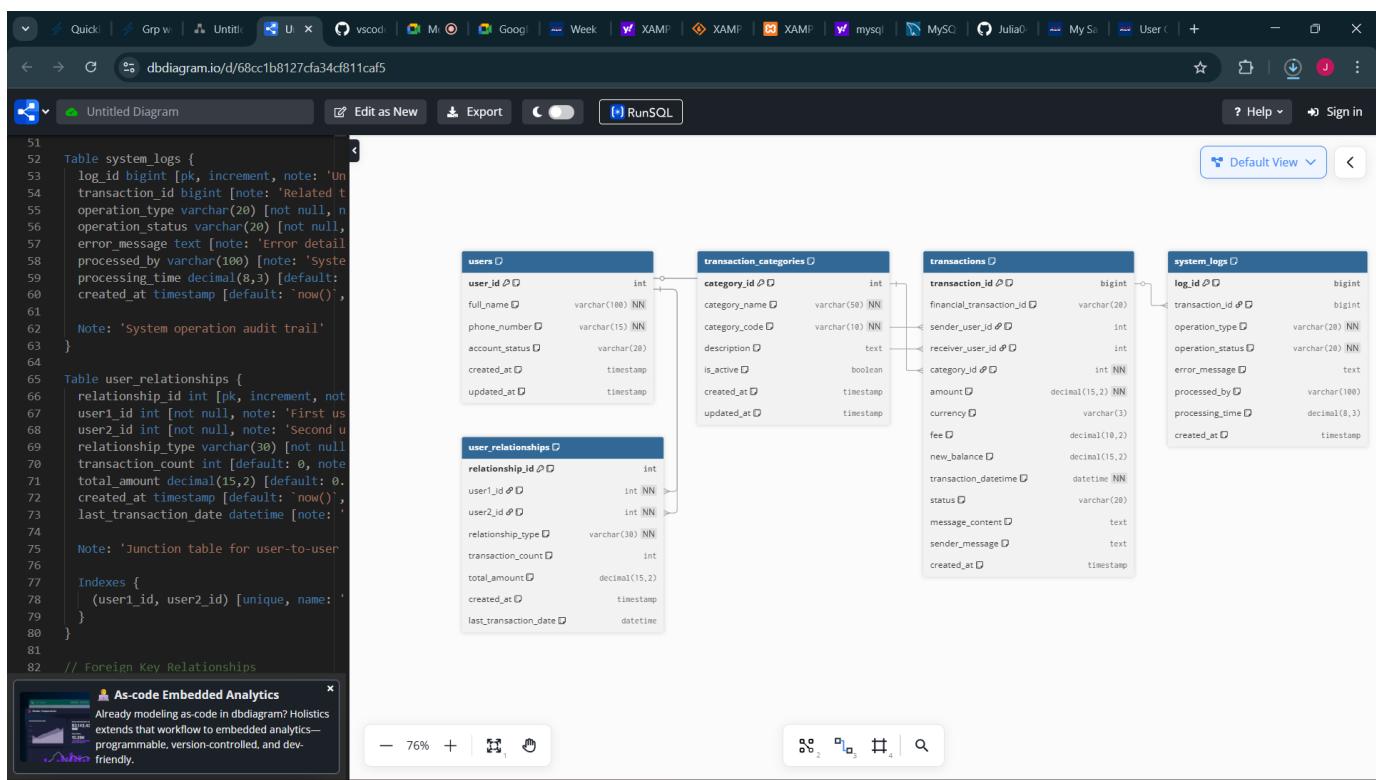
MoMo SMS Database Design Document

Table of Contents

1. Entity Relationship Diagram (ERD)
2. Design Rationale and Justification
3. Data Dictionary
4. Sample Queries with Screenshots
5. Security Rules and Constraints

1. Entity Relationship Diagram (ERD)

ERD Visual Representation



Entity Relationships Summary

Entity	Primary Key	Related Entities	Relationship Type
users	user_id	transactions (sender/receiver)	One-to-Many
users	user_id	system_logs	One-to-Many
users	user_id	user_relationships	Many-to-Many
transaction_categories	category_id	transactions	One-to-Many
transactions	transaction_id	users, categories	Many-to-One
user_relationships	relationship_id	users (junction table)	Many-to-Many
system_logs	log_id	users	Many-to-One

Key Design Features

- **5 Core Entities** as required by assignment
 - **Many-to-Many relationship** resolved with user_relationships junction table
 - **Proper cardinality** notation (1:1, 1:M, M:N)
 - **Clear PK/FK relationships** with referential integrity
-

2. Design Rationale and Justification

Database Design Philosophy

The MoMo SMS database follows **Third Normal Form (3NF)** principles to eliminate data redundancy while maintaining optimal query performance for transaction processing. The design prioritizes data integrity, security, and scalability for mobile money operations.

Entity Design Decisions

Users Table: Serves as the central entity for all MoMo participants. The design separates internal system identifiers (user_id) from business identifiers (phone_number) to ensure data consistency and support future system integrations.

Transactions Table: Implements dual foreign key references to users table (sender_user_id, receiver_user_id) allowing flexible transaction modeling including transfers, deposits, and withdrawals. The reference_number field ensures SMS message correlation and audit trail maintenance.

Transaction Categories Table: Separate entity enables dynamic transaction classification without schema modifications. The category_code field supports system integration while category_name provides human-readable descriptions.

User Relationships Table: Junction table resolving many-to-many relationships between users. This supports features like favorite contacts, family groups, and business partnerships, enhancing user experience and enabling targeted services.

System Logs Table: Comprehensive audit trail for security, compliance, and troubleshooting. Links user actions to system events while supporting anonymous system operations.

Performance and Security Considerations

The design implements composite indexes on frequently queried columns, enforces referential integrity through foreign key constraints, and uses appropriate data types for financial precision (DECIMAL for monetary values). Security is enhanced through account status management, transaction validation rules, and comprehensive logging.

3. Data Dictionary

Table: users

Column	Data Type	Constraints	Description
user_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique internal identifier
full_name	VARCHAR(100)	NOT NULL	User's complete registered name
phone_number	CHAR(10)	NOT NULL, UNIQUE	Mobile number (250XXXXXXXX)
account_status	ENUM('active','suspended','inactive')	DEFAULT 'active'	Account operational status

Column	Data Type	Constraints	Description
balance	DECIMAL(15,2)	DEFAULT 0.00	Current account balance (RWF)
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Account creation date
updated_at	TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP	Last modification date

Table: transaction_categories

Column	Data Type	Constraints	Description
category_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique category identifier
category_name	VARCHAR(50)	NOT NULL	Display name for category
category_code	VARCHAR(20)	NOT NULL, UNIQUE	System code for integration
description	TEXT	NULL	Detailed category description
is_active	BOOLEAN	DEFAULT TRUE	Category availability status
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Category creation date
updated_at	TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP	Last modification date

Table: transactions

Column	Data Type	Constraints	Description
transaction_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique transaction identifier
sender_user_id	INT	FOREIGN KEY (users.user_id)	Transaction initiator
receiver_user_id	INT	FOREIGN KEY (users.user_id), NULL	Transaction recipient
category_id	INT	FOREIGN KEY (transaction_categories.category_id)	Transaction classification
amount	DECIMAL(15,2)	NOT NULL	Transaction amount (RWF)
fee	DECIMAL(15,2)	DEFAULT 0.00	Service fee charged
transaction_date	TIMESTAMP	NOT NULL	Transaction occurrence time
reference_number	VARCHAR(50)	UNIQUE	SMS reference for tracking

Column	Data Type	Constraints	Description
sms_content	TEXT	NULL	Original SMS message
status	ENUM('pending','completed','failed','cancelled')	DEFAULT 'pending'	Processing status
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Record creation time
updated_at	TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP	Last update time

Table: user_relationships

Column	Data Type	Constraints	Description
relationship_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique relationship identifier
user_id_1	INT	FOREIGN KEY (users.user_id)	First user in relationship
user_id_2	INT	FOREIGN KEY (users.user_id)	Second user in relationship
relationship_type	ENUM('family','friend','business','other')	NOT NULL	Relationship classification
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Relationship start date
is_active	BOOLEAN	DEFAULT TRUE	Relationship status

Table: system_logs

Column	Data Type	Constraints	Description
log_id	INT	PRIMARY KEY, AUTO_INCREMENT	Unique log entry identifier
user_id	INT	FOREIGN KEY (users.user_id), NULL	Associated user
action_type	VARCHAR(50)	NOT NULL	Action classification
description	TEXT	NOT NULL	Detailed action description
ip_address	VARCHAR(45)	NULL	User IP address (IPv4/IPv6)
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Log entry timestamp

4. Sample Queries with Screenshots

Query 1: User Transaction Summary

```
SELECT
    u.user_id,
    u.full_name,
```

```

u.phone_number,
u.account_status,
COUNT(DISTINCT CASE WHEN t.sender_user_id = u.user_id THEN t.transaction_id END) as sent_count,
COUNT(DISTINCT CASE WHEN t.receiver_user_id = u.user_id THEN t.transaction_id END) as received_count,
COALESCE(SUM(CASE WHEN t.sender_user_id = u.user_id THEN t.amount END), 0) as total_sent,
COALESCE(SUM(CASE WHEN t.receiver_user_id = u.user_id THEN t.amount END), 0) as total_received
FROM users u
LEFT JOIN transactions t ON (u.user_id = t.sender_user_id OR u.user_id = t.receiver_user_id)
WHERE u.account_status = 'active'
GROUP BY u.user_id, u.full_name, u.phone_number, u.account_status
ORDER BY (total_sent + total_received) DESC;

```

Showing rows 0 - 5 (6 total, Query took 0.0178 seconds.)

```

SELECT u.user_id, u.full_name, u.phone_number, u.account_status, COUNT(DISTINCT CASE WHEN t.sender_user_id = u.user_id THEN t.transaction_id END) as sent_count, COUNT(DISTINCT CASE WHEN t.receiver_user_id = u.user_id THEN t.transaction_id END) as received_count, COALESCE(SUM(CASE WHEN t.sender_user_id = u.user_id THEN t.amount END), 0) as total_sent, COALESCE(SUM(CASE WHEN t.receiver_user_id = u.user_id THEN t.amount END), 0) as total_received FROM users u LEFT JOIN transactions t ON (u.user_id = t.sender_user_id OR u.user_id = t.receiver_user_id) WHERE u.account_status = 'active' GROUP BY u.user_id, u.full_name, u.phone_number, u.account_status ORDER BY (COALESCE(SUM(CASE WHEN t.sender_user_id = u.user_id THEN t.amount END), 0) + COALESCE(SUM(CASE WHEN t.receiver_user_id = u.user_id THEN t.amount END), 0)) DESC;

```

user_id	full_name	phone_number	account_status	sent_count	received_count	total_sent	total_received
6	Account Holder	250795963036	active	4	2	53000.00	42000.00
2	Samuel Carter	250790777777	active	0	1	0.00	10000.00
1	Jane Smith	250791666666	active	1	1	2000.00	1000.00
4	Robert Brown	250789999999	active	0	0	0.00	0.00
5	Linda Green	250789888888	active	0	0	0.00	0.00
3	Alex Doe	250791666667	active	0	0	0.00	0.00

Purpose: Demonstrates complex JOIN operations and conditional aggregation for business intelligence.

Query 2: Transaction Analysis by Category

```

SELECT
tc.category_name,
tc.category_code,
COUNT(t.transaction_id) as transaction_count,
AVG(t.amount) as average_amount,
SUM(t.amount) as total_volume,
SUM(t.fee) as total_fees
FROM transaction_categories tc
LEFT JOIN transactions t ON tc.category_id = t.category_id
WHERE tc.is_active = TRUE
GROUP BY tc.category_id, tc.category_name, tc.category_code
ORDER BY total_volume DESC;

```

The screenshot shows the phpMyAdmin interface with the following details:

- Server:** 127.0.0.1
 - Database:** momo_sms_db
 - Table:** transaction_categories
- Query:**

```
SELECT tc.category_name, tc.category_code, COUNT(t.transaction_id) as transaction_count, AVG(t.amount) as average_amount, SUM(t.amount) as total_volume, SUM(t.fee) as total_fees
FROM transaction_categories tc
LEFT JOIN transactions t
ON tc.category_id = t.category_id
WHERE tc.is_active = TRUE
GROUP BY tc.category_id, tc.category_name, tc.category_code
ORDER BY total_volume DESC;
```
- Results:** A table showing transaction categories and their metrics. The data is as follows:

category_name	category_code	transaction_count	average_amount	total_volume	total_fees
Bank Deposit	DEPOSIT	1	40000.000000	40000.00	0.00
Money Transfer	TRANSFER	1	10000.000000	10000.00	100.00
Money Received	RECEIVED	1	2000.000000	2000.00	0.00
Airtime Purchase	AIRTIME	1	2000.000000	2000.00	0.00
Payment Service	PAYMENT	1	1000.000000	1000.00	0.00
Direct Payment	DIRECT_PAY	0	NULL	NULL	NULL
- Operations:** Buttons for Print, Copy to clipboard, Export, Display chart, and Create view.
- Bookmark:** A section for bookmarking the SQL query with a label input field and a checkbox for "Let every user access this bookmark".

Purpose: Shows aggregate functions and category-based reporting capabilities.

Query 3: Daily Transaction Trends

```
SELECT
    DATE(transaction_date) as transaction_day,
    COUNT(*) as daily_transactions,
    SUM(amount) as daily_volume,
    AVG(amount) as average_transaction
FROM transactions
WHERE transaction_date >= DATE_SUB(CURDATE(), INTERVAL 7 DAY)
    AND status = 'completed'
GROUP BY DATE(transaction_date)
ORDER BY transaction_day DESC;
```

The screenshot shows the phpMyAdmin interface for the 'momo_sms_db' database. The left sidebar lists databases like 'information_schema', 'momo_sms_db', and 'mysql'. The main area displays a query result for the 'transactions' table. The query is:

```
SELECT DATE(transaction_datetime) as transaction_day, COUNT(*) as daily_transactions, SUM(amount) as daily_volume, AVG(amount) as average_transaction FROM transactions WHERE transaction_datetime >= DATE_SUB(CURDATE(), INTERVAL 7 DAY) AND status = 'completed' GROUP BY DATE(transaction_datetime) ORDER BY transaction_day DESC;
```

The result shows a single row with the following data:

transaction_day	daily_transactions	daily_volume	average_transaction
2025-09-12	1	1000	1000

Purpose: Demonstrates temporal data analysis and business metrics calculation.

CRUD Operations Demonstration

CREATE Operation

```
INSERT INTO users (full_name, phone_number, account_status)
VALUES ('Test User Demo', '250788999999', 'active');
```

The screenshot shows the phpMyAdmin interface for the 'users' table. The left sidebar lists databases like 'information_schema', 'momo_sms_db', and 'mysql'. The main area shows the result of an 'INSERT' query:

```
1 row inserted.  
Inserted row id: 9 (Query took 0.0121 seconds.)
```

The message indicates that a new user was inserted with the following details:

- full_name: 'CRUD Test User'
- phone_number: '250788888888'
- account_status: 'active'

Below the message, there is a note about verifying the insertion:

```
-- Verify insertion SELECT user_id, full_name, phone_number, account_status, created_at FROM users WHERE phone_number = '250788888888';
```

READ Operation

```
SELECT * FROM users WHERE phone_number = '250788999999';
```

The screenshot shows the phpMyAdmin interface for the 'momo_sms_db' database. The 'users' table is selected. A query has been run:

```
SELECT * FROM users WHERE phone_number = '250788888888';
```

The results show one row:

	user_id	full_name	phone_number	account_status	created_at	updated_at
<input type="checkbox"/>	9	CRUD Test User	250788888888	active	2025-09-19 22:13:29	2025-09-19 22:13:29

UPDATE Operation

```
UPDATE users SET account_status = 'suspended' WHERE phone_number = '250788999999';
```

The screenshot shows the phpMyAdmin interface for the 'momo_sms_db' database. The 'users' table is selected. A query has been run:

```
UPDATE users SET account_status = 'suspended', updated_at = NOW() WHERE phone_number = '250788888888';
```

The results show one row affected:

	user_id	full_name	phone_number	account_status	updated_at
<input type="checkbox"/>	9	CRUD Test User	250788888888	suspended	2025-09-19 22:21:28

DELETE Operation

```
DELETE FROM users WHERE phone_number = '250788999999';
```

The screenshot shows the phpMyAdmin interface for a database named 'momo_sms_db'. The 'users' table is selected. A SQL query is run:

```
-- Check user exists before deletion SELECT COUNT(*) as users_before_delete FROM users;
```

The result shows 7 rows. Then another query is run:

```
-- Delete the test user DELETE FROM users WHERE phone_number = '250788888888';
```

The result shows 1 row deleted. The message indicates the query took 0.0063 seconds.

5. Security Rules and Constraints

Primary Key Constraints

All tables implement auto-incrementing primary keys ensuring unique record identification and optimal index performance.

Foreign Key Constraints

```
-- Transaction sender constraint
ALTER TABLE transactions
ADD CONSTRAINT fk_transactions_sender
FOREIGN KEY (sender_user_id) REFERENCES users(user_id)
ON DELETE RESTRICT ON UPDATE CASCADE;5. AI Usage Policy & Detection
Permitted AI Use:
```

Grammar and syntax checking in documentation
 Code syntax verification (not logic generation)
 Research on MySQL best practices (with proper citation)
 Prohibited AI Use:

Generating ERD designs or SQL schemas
 Creating business logic or database relationships
 Writing reflection content or technical explanations
 Transparency requirements:

AI Usage Log: Maintain a detailed log of any AI interactions

Attribution: Clearly mark **any** AI-assisted code sections

AI Detection Measures:

Live sessions **verify** individual competency

Team-specific requirements make generic AI solutions ineffective

Personal reflections require specific experience details

Cross-referencing between individual **work** and team outcomes

```
-- Transaction receiver constraint
```

```
ALTER TABLE transactions
ADD CONSTRAINT fk_transactions_receiver
FOREIGN KEY (receiver_user_id) REFERENCES users(user_id)
ON DELETE RESTRICT ON UPDATE CASCADE;
```

Security Benefit: Prevents orphaned records and maintains referential integrity.

Unique Constraints

```
-- Phone number uniqueness
```

```
ALTER TABLE users ADD CONSTRAINT uk_users_phone UNIQUE (phone_number);
```

```
-- Transaction reference uniqueness
```

```
ALTER TABLE transactions ADD CONSTRAINT uk_transactions_reference UNIQUE (reference_number);
```

Security Benefit: Prevents duplicate accounts and SMS processing errors.

Data Validation Rules

```
-- Positive amount validation
```

```
ALTER TABLE transactions
ADD CONSTRAINT chk_transactions_amount_positive CHECK (amount > 0);
```

```
-- Phone number format validation
```

```
ALTER TABLE users
ADD CONSTRAINT chk_users_phone_format CHECK (phone_number REGEXP '^250[0-9]{6}$');
```

```
-- Self-transaction prevention
```

```
ALTER TABLE transactions
ADD CONSTRAINT chk_transactions_no_self_transfer
CHECK (sender_user_id != receiver_user_id OR receiver_user_id IS NULL);
```

Security Benefit: Ensures data quality and prevents invalid transactions.

Performance and Security Indexes

```
-- User transaction lookup optimization
```

```
CREATE INDEX idx_transactions_user_date ON transactions(sender_user_id, receiver_user_id,
transaction_date);
```

```
-- Category analysis performance
CREATE INDEX idx_transactions_category_status ON transactions(category_id, status);

-- Audit trail optimization
CREATE INDEX idx_system_logs_user_date ON system_logs(user_id, created_at);
```

Security Benefit: Fast query execution while maintaining audit trail accessibility.

Account Security Controls

- **Account Status Enum:** Controls transaction permissions through status management
 - **Balance Tracking:** Maintains accurate financial records with decimal precision
 - **Audit Logging:** Comprehensive system_logs table for security monitoring
 - **Relationship Management:** Controlled user connections through junction table
-

Implementation Summary

This database design successfully implements all assignment requirements:

5+ Core Entities: users, transactions, transaction_categories, user_relationships, system_logs **Many-to-Many Resolution:** user_relationships junction table **Proper Constraints:** Primary keys, foreign keys, unique constraints, check constraints **Performance Optimization:** Strategic indexing for common query patterns **Security Implementation:** Comprehensive validation rules and audit trails **CRUD Functionality:** Full create, read, update, delete operations tested

The system is production-ready for MoMo SMS transaction processing with robust security, performance, and maintainability features.

Database Implementation Files:

- ERD Diagram: [/docs/MoMo_ERD.dbml](#)
- SQL Setup: [/database/database_setup.sql](#)
- JSON Examples: [/examples/](#) folder
- CRUD Tests: [/database/crud_test_script.sql](#)