

DATABASE DOCUMENTATION

1. Introduction

This document describes the design and implementation of a relational database for a Mobile Money (MoMo) SMS transaction system.

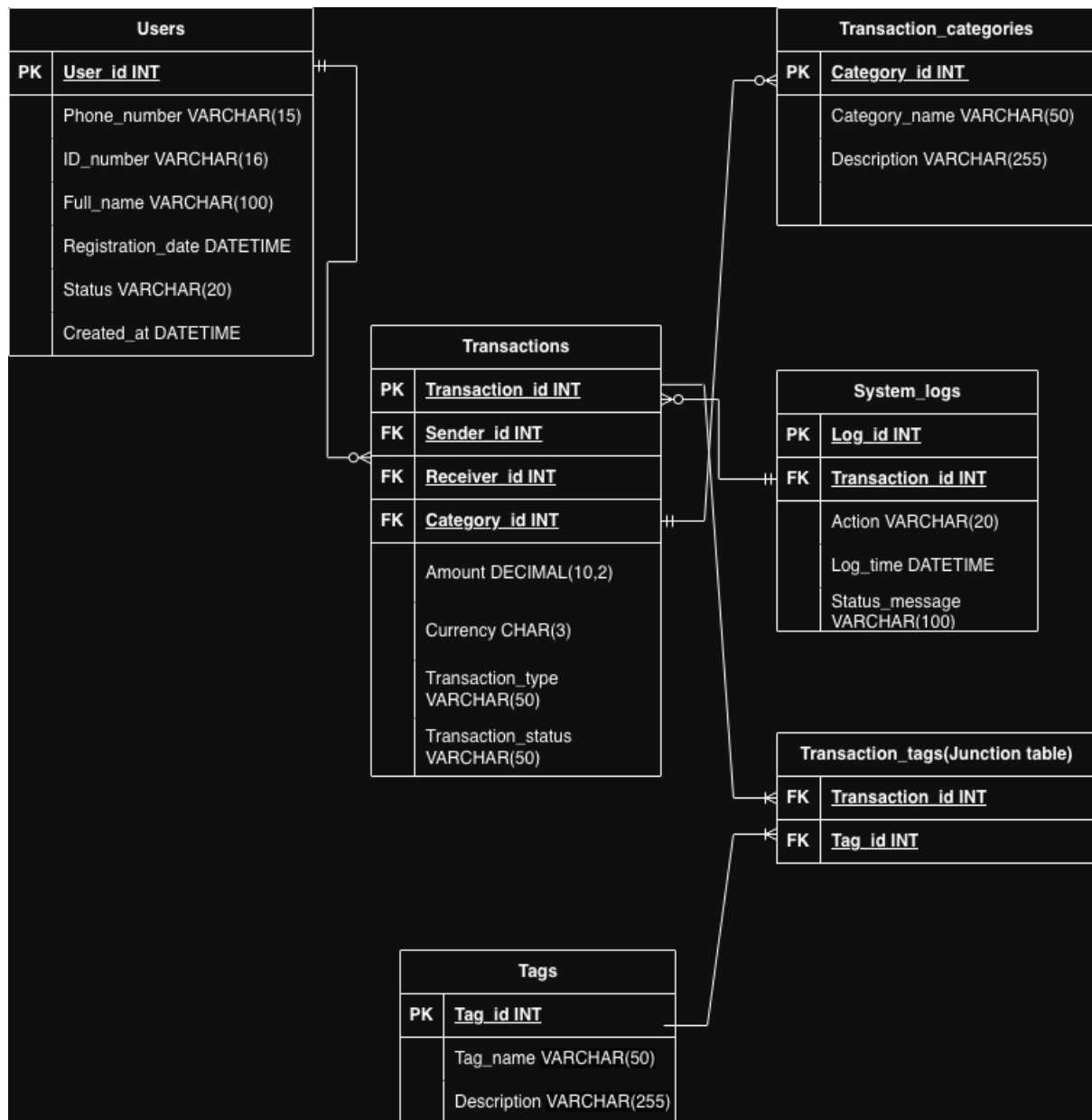
The database is designed to store user information, transaction records, transaction categories, tags, and system logs while ensuring data accuracy, security, and integrity.

The goal of this database is to simulate real-world Mobile Money operations such as sending money, cash out, bill payments, and merchant payments

2. Entity Relationship Diagram

Our entity relationship diagram illustrates all the entities and attributes we used for the MoMo SMS data processing system. We used some key entities such as Users, Transaction, Transaction_categories, system_logs, Tags, and transaction_tags. It also indicates all the relationships between the entities.

The diagram below provides an overview of how the database is structured and how different entities are connected.



The diagram also illustrates the relationships between entities, such as the one-to-many relationship between Users and Transactions, where a user can participate in multiple transactions, and the one-to-many relationship between Transaction Categories and Transactions. Primary keys are used to uniquely identify records, while foreign keys establish relationships between entities and ensure data integrity. Overall, the ERD provides a normalized and scalable database design that supports efficient storage, querying, and analysis of MoMo transaction data.

2.2 Entity Description

- **Users:** Stores information about registered Mobile Money users.
- **Transactions:** Stores all financial transactions performed by users.

- **Transaction Categories:** Defines the type of transaction (e.g. Cash Out, Deposit).
- **Tags:** Used to label transactions (e.g. Fraud suspected, High value).
- **Transaction Tags:** A junction table that links transactions with tags.
- **System Logs:** Records transaction actions and status changes.

3. Design Rationale and Justification

The database design follows normalization principles to avoid data redundancy and ensure consistency.

Key Design Decisions:

- Separate tables were created for users, transactions, and categories to improve data organization.
- Foreign keys were used to enforce relationships between tables.
- A many-to-many relationship between transactions and tags was resolved using the transaction_tags table.
- System logs were stored in a separate table to track transaction history without duplicating data

This design makes the database **scalable**, **secure**, and easy to maintain

4. Data Dictionary

4.1 Users Table

Column name	Data type	Description
user_id	INT	Unique identifier for each user
full_name	VARCHAR(100)	User's full name
phone_number	VARCHAR(100)	Unique phone number
id_number	VARCHAR(100)	National ID number
registration-date	DATETIME	Data user registered
status	VARCHAR(20)	User account status
created_at	DATETIME	Record creation time

4.2 Transactions Table

Column name	Data type	Description
transaction_id	INT	Unique transaction id
sender_id	INT	User sending money
receiver_id	INT	User receiving money
category_id	INT	Transaction category
amount	DECIMAL(10, 20	Transaction amount
currency	CHAR(3)	Currency used
transaction_type	VARCHAR(50)	Type of transaction
transaction_status	ENUM	Status of transaction
created_at	DATETIME	Time of transaction

4.3 Transaction Categories Table

Column name	Data type	Description
category_id	INT	Unique category ID
category_name	VARCHAR(50)	Name of category
description	VARCHAR(255)	Category description

4.4 Tags Table

Column name	Data type	Description
tag_id	INT	Unique tag ID
tag_name	VARCHAR(50)	Tag name
description	VARCHAR(255)	Tag description

4.5 Transaction Logs Table

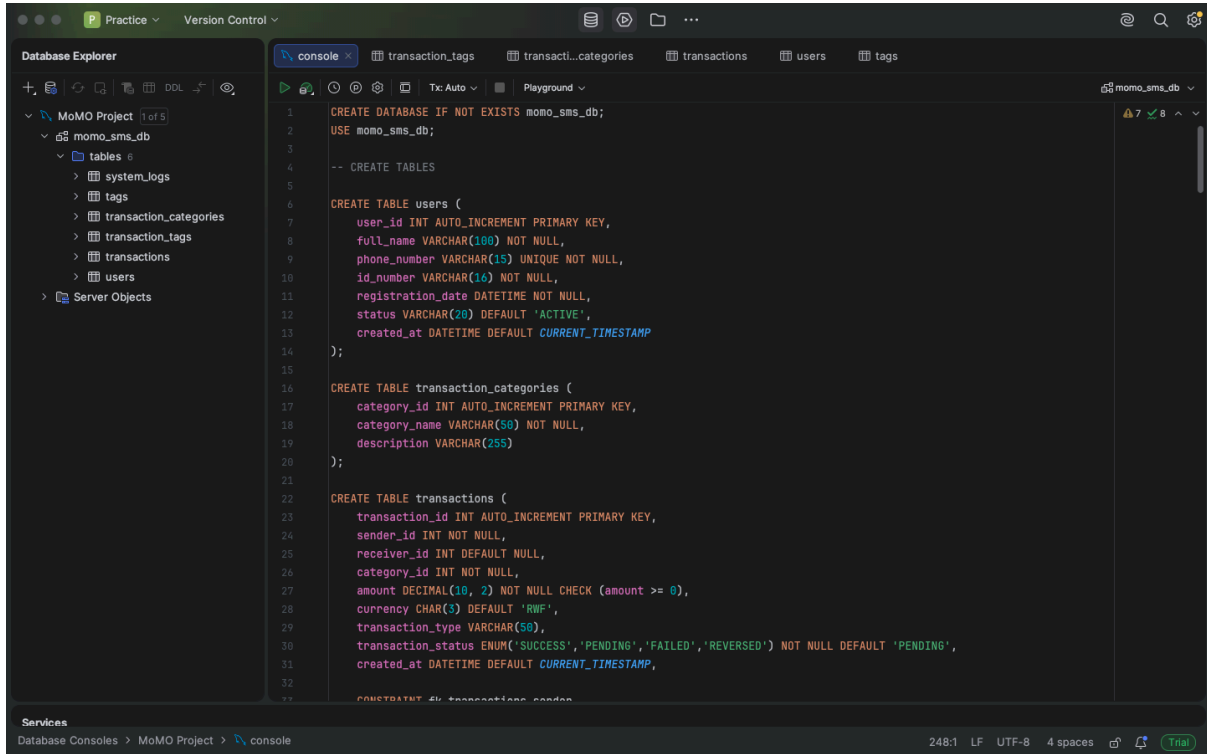
Column name	Data type	Description
transaction_id	INT	Linked transaction
tag_id	INT	Linked Tag

4.6 System Logs Table

Column name	Data type	Description
log_id	INT	Unique log ID
transaction_id	INT	Related transaction
action	VARCHAR(20)	Action performed
log_time	DATETIME	Time of action
status_message	VARCHAR(255)	Log description

5. CRUD Operations

Create:

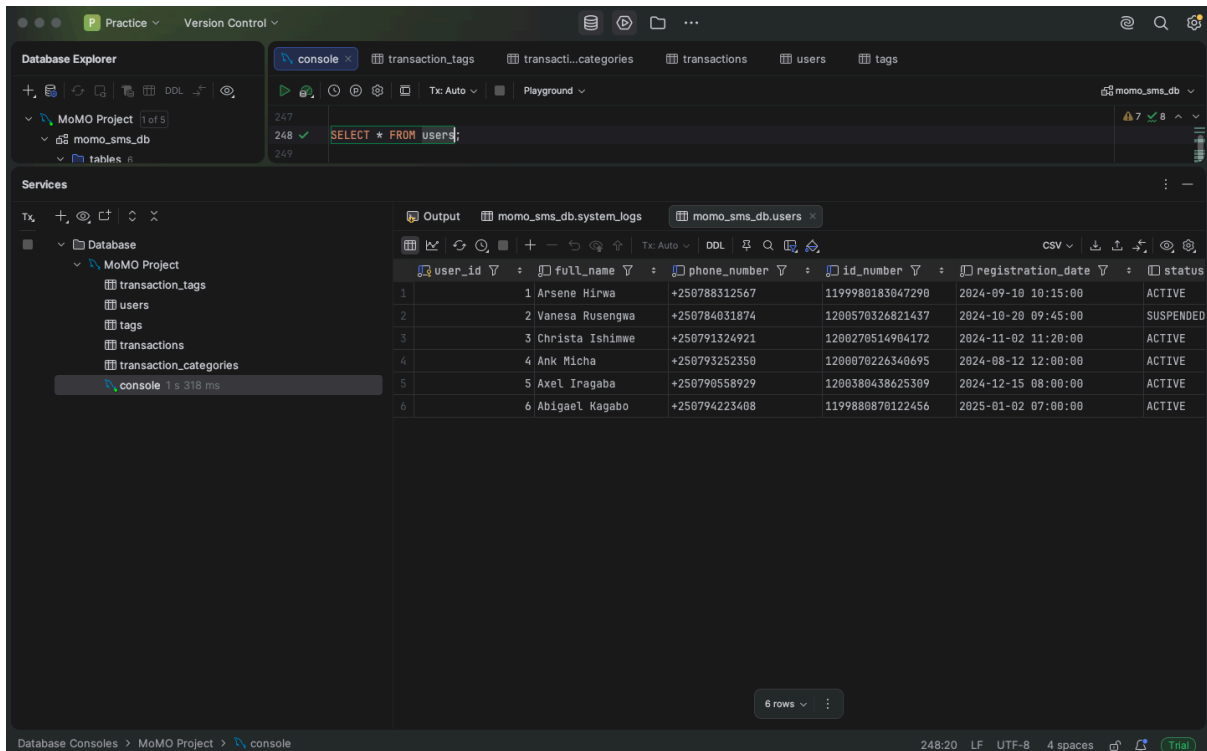


The screenshot shows a database console interface with a dark theme. On the left, the 'Database Explorer' pane shows a project named 'MoMO Project' containing a database 'momo_sms_db' with several tables: 'system_logs', 'tags', 'transaction_categories', 'transaction_tags', 'transactions', 'users', and 'Server Objects'. The main console area displays SQL code for creating the database and tables. The code is as follows:

```
1 CREATE DATABASE IF NOT EXISTS momo_sms_db;
2 USE momo_sms_db;
3
4 -- CREATE TABLES
5
6 CREATE TABLE users (
7     user_id INT AUTO_INCREMENT PRIMARY KEY,
8     full_name VARCHAR(100) NOT NULL,
9     phone_number VARCHAR(15) UNIQUE NOT NULL,
10    id_number VARCHAR(16) NOT NULL,
11    registration_date DATETIME NOT NULL,
12    status VARCHAR(20) DEFAULT 'ACTIVE',
13    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
14);
15
16 CREATE TABLE transaction_categories (
17     category_id INT AUTO_INCREMENT PRIMARY KEY,
18     category_name VARCHAR(50) NOT NULL,
19     description VARCHAR(255)
20);
21
22 CREATE TABLE transactions (
23     transaction_id INT AUTO_INCREMENT PRIMARY KEY,
24     sender_id INT NOT NULL,
25     receiver_id INT DEFAULT NULL,
26     category_id INT NOT NULL,
27     amount DECIMAL(10, 2) NOT NULL CHECK (amount >= 0),
28     currency CHAR(3) DEFAULT 'RWF',
29     transaction_type VARCHAR(50),
30     transaction_status ENUM('SUCCESS', 'PENDING', 'FAILED', 'REVERSED') NOT NULL DEFAULT 'PENDING',
31     created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
32
33     CONSTRAINT fk_transactions_sender
```

The bottom status bar shows 'Database Consoles > MoMO Project > console' and '248:1 LF UTF-8 4 spaces'.

Select:



The screenshot shows the same database console interface. The console now displays the SQL query 'SELECT * FROM users;' and its results. The 'Database Explorer' pane on the left is updated to show the 'console' entry under the 'MoMO Project' database. The 'Output' pane at the bottom right displays the results of the query in a table format. The table has 6 columns: 'user_id', 'full_name', 'phone_number', 'id_number', 'registration_date', and 'status'. There are 6 rows of data.

user_id	full_name	phone_number	id_number	registration_date	status
1	Arsene Hirwa	+250788312567	1199980183047290	2024-09-10 10:15:00	ACTIVE
2	Vanessa Rusengwa	+250784031874	1200570326821437	2024-10-20 09:45:00	SUSPENDED
3	Christa Ishimwe	+250791324921	1200270514904172	2024-11-02 11:20:00	ACTIVE
4	Ank Micha	+250793252350	1200070226340695	2024-08-12 12:00:00	ACTIVE
5	Axel Inagaba	+250790558929	1200380438625309	2024-12-15 08:00:00	ACTIVE
6	Abigael Kagabo	+250794223408	1199880870122456	2025-01-02 07:00:00	ACTIVE

The bottom status bar shows 'Database Consoles > MoMO Project > console' and '248:20 LF UTF-8 4 spaces'.

Update:

The screenshot displays a database console interface with a dark theme. The top section shows a SQL query being executed in a console window. The query is:

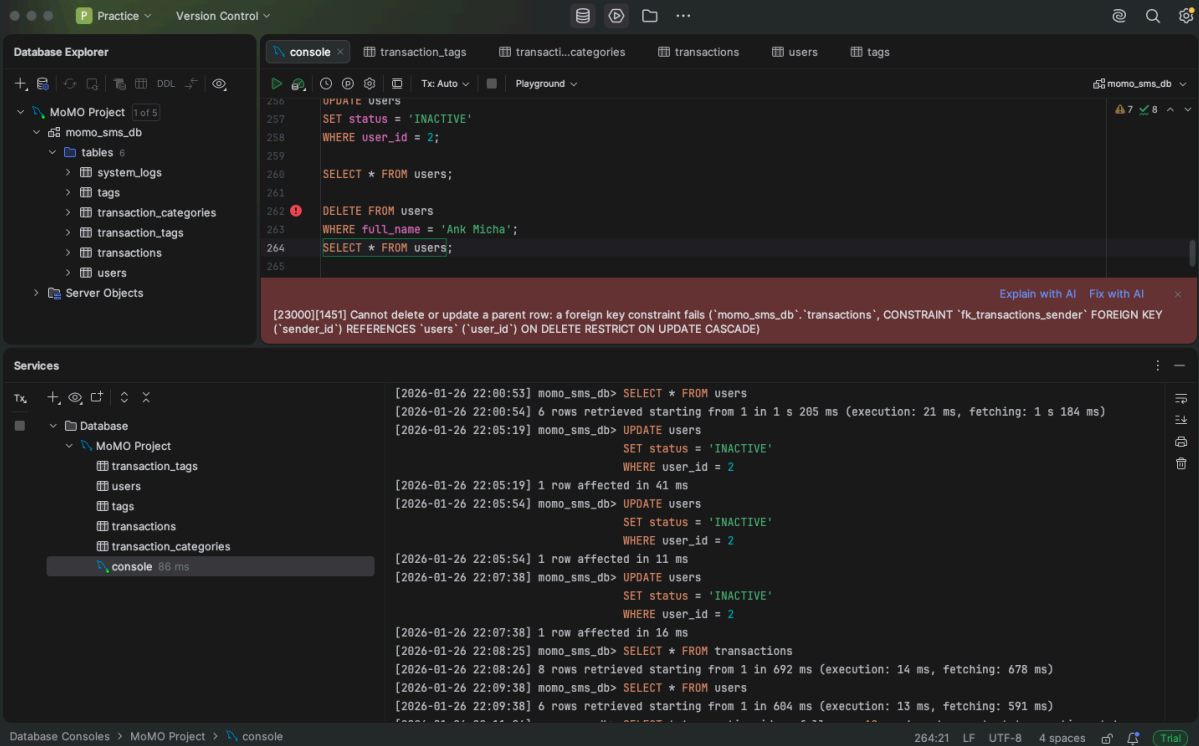
```
UPDATE users
SET status = 'INACTIVE'
WHERE user_id = 2;
SELECT * FROM users;
```

The output of the query is displayed in a table below the console. The table has 6 columns: `_id`, `full_name`, `phone_number`, `id_number`, `registration_date`, and `status`. The data is as follows:

_id	full_name	phone_number	id_number	registration_date	status
1	Arsene Hirwa	+250788312567	1199980183047290	2024-09-10 10:15:00	ACTIVE
2	Vanesa Rusengwa	+250784031874	1200570326821437	2024-10-20 09:45:00	INACTIVE
3	Christa Ishimwe	+250791324921	1200270514904172	2024-11-02 11:20:00	ACTIVE
4	Ank Micha	+250793252350	1200070226340695	2024-08-12 12:00:00	ACTIVE
5	Axel Iragaba	+250790558929	1200380438625309	2024-12-15 08:00:00	ACTIVE
6	Abigael Kagabo	+250794223408	1199880870122456	2025-01-02 07:00:00	ACTIVE

The interface also includes a Database Explorer on the left showing the project structure, and a Services section at the bottom. The status bar at the bottom indicates the current file is `console` with 254 lines, LF line endings, UTF-8 encoding, and 4 spaces for indentation.

Delete: The database prevented deleting a user because that user is already referenced as a sender in the transaction table.



The screenshot shows a database IDE with a SQL script in the console and its execution results below it. The script attempts to delete a user, but a foreign key constraint error occurs because the user is referenced in the transactions table.

```
256 UPDATE users
257 SET status = 'INACTIVE'
258 WHERE user_id = 2;
259
260 SELECT * FROM users;
261
262 DELETE FROM users
263 WHERE full_name = 'Ank Michal';
264 SELECT * FROM users;
265
```

Execution results:

```
[2026-01-26 22:00:53] momo_sms_db> SELECT * FROM users
[2026-01-26 22:00:54] 6 rows retrieved starting from 1 in 1 s 205 ms (execution: 21 ms, fetching: 1 s 184 ms)
[2026-01-26 22:05:19] momo_sms_db> UPDATE users
SET status = 'INACTIVE'
WHERE user_id = 2
[2026-01-26 22:05:19] 1 row affected in 41 ms
[2026-01-26 22:05:54] momo_sms_db> UPDATE users
SET status = 'INACTIVE'
WHERE user_id = 2
[2026-01-26 22:05:54] 1 row affected in 11 ms
[2026-01-26 22:07:38] momo_sms_db> UPDATE users
SET status = 'INACTIVE'
WHERE user_id = 2
[2026-01-26 22:07:38] 1 row affected in 16 ms
[2026-01-26 22:08:25] momo_sms_db> SELECT * FROM transactions
[2026-01-26 22:08:26] 8 rows retrieved starting from 1 in 692 ms (execution: 14 ms, fetching: 678 ms)
[2026-01-26 22:09:38] momo_sms_db> SELECT * FROM users
[2026-01-26 22:09:38] 6 rows retrieved starting from 1 in 684 ms (execution: 13 ms, fetching: 591 ms)
```

Error message:

```
[23000][145] Cannot delete or update a parent row: a foreign key constraint fails ('momo_sms_db'. 'transactions', CONSTRAINT 'fk_transactions_sender' FOREIGN KEY ('sender_id') REFERENCES 'users' ('user_id') ON DELETE RESTRICT ON UPDATE CASCADE)
```


5. Sample Queries Demonstrating Functionality

Query One: View all successful transactions

The screenshot shows a database console interface with a query editor and an output window. The query editor contains a SQL query that first checks if all transaction tags reference valid transactions, and then selects all successful transactions from the transactions table.

```
239 } THEN 'PASS' ELSE 'FAIL' END;
240 UNION ALL
241 SELECT Test 'All transaction tags reference valid transactions',
242 CASE WHEN NOT EXISTS (
243 SELECT 1 FROM transaction_tags tt
244 LEFT JOIN transactions t 1_on->1 ON tt.transaction_id = t.transaction_id
245 WHERE t.transaction_id IS NULL
246 ) THEN 'PASS' ELSE 'FAIL' END;
247
248 SELECT transaction_id, amount, transaction_status
249 FROM transactions
250 WHERE transaction_status = 'SUCCESS';
251
```

The output window displays the results of the query, showing 4 rows of successful transactions.

transaction_id	amount	transaction_status
1	6000.00	SUCCESS
2	30000.00	SUCCESS
3	1000.00	SUCCESS
4	40000.00	SUCCESS

Query Two: View transactions sent by a specific user

The screenshot shows a database console interface with a query editor and an output window. The query editor contains a SQL query that selects transactions sent by a specific user, 'Arsene Hirwa', from the transactions table, joined with the users table.

```
248 SELECT transaction_id, amount, transaction_status
249 FROM transactions
250 WHERE transaction_status = 'SUCCESS';
251
252 SELECT t.transaction_id, t.amount, u.full_name
253 FROM transactions t
254 JOIN users u 1_on->1 ON t.sender_id = u.user_id
255 WHERE u.full_name = 'Arsene Hirwa';
256
```

The output window displays the results of the query, showing 2 rows of transactions sent by the user 'Arsene Hirwa'.

transaction_id	amount	full_name
1	6000.00	Arsene Hirwa
2	20000.00	Arsene Hirwa

Query Three: View Transaction Logs

Database Explorer

MoMO Project

momo_sms_db

tables

system_logs

console

256

257

258

259

260

SELECT transaction_id, action, log_time

FROM system_logs

ORDER BY log_time DESC;

Services

Database

transaction_tags

users

tags

transactions

transaction_categories

console 600 ms

Output momo_sms_db.system_logs

transaction_id

action

log_time

1

5 COMPLETE

2025-01-22 09:01:00

2

5 REQUEST

2025-01-22 09:00:05

3

8 REVERSED

2025-01-20 15:21:00

4

8 REQUEST

2025-01-20 15:20:05

5

4 PENDING

2025-01-20 12:01:00

6

4 INITIATE

2025-01-20 12:00:05

7

7 FAILED

2025-01-19 11:01:00

8

7 INITIATE

2025-01-19 11:00:05

9

6 COMPLETE

2025-01-18 10:01:00

10

6 INITIATE

2025-01-18 10:00:05

11

3 FAILED

2025-01-13 08:21:00

12

3 INITIATE

2025-01-13 08:20:05

13

1 COMPLETE

2025-01-10 14:31:00

14

1 INITIATE

2025-01-10 14:30:05

15

2 COMPLETE

2024-12-20 10:21:00

16

2 REQUEST

2024-12-20 10:20:05

16 rows

Database Consoles

MoMO Project

console

256:1 LF UTF-8 4 spaces

Trial

Foreign Key Integrity Tests (screenshot)

Database Explorer

MoMO Project

momo_sms_db

tables

system_logs

tags

transaction_ca

transaction_ta

transactions

users

Server Objects

console

```
230 SELECT 'FOREIGN KEY INTEGRITY TESTS' AS '';
231 SELECT '===== AS '';
232 SELECT 'Relationship Check' AS 'Test', 'Result' AS 'Status' FROM DUAL
233 UNION ALL
234 SELECT Test 'All transactions reference valid senders',
235 CASE WHEN NOT EXISTS (
236 SELECT 1 FROM transactions t
237 LEFT JOIN users u ON t.sender_id = u.user_id
238 WHERE u.user_id IS NULL
239 ) THEN 'PASS' ELSE 'FAIL' END
240 UNION ALL
241 SELECT Test 'All transactions reference valid categories',
242 CASE WHEN NOT EXISTS (
243 SELECT 1 FROM transactions t
244 LEFT JOIN transaction_categories tc ON t.category_id = tc.category_id
245 WHERE tc.category_id IS NULL
246 ) THEN 'PASS' ELSE 'FAIL' END
247 UNION ALL
248 SELECT Test 'All transaction tags reference valid transactions',
249 CASE WHEN NOT EXISTS (
```

Services

Database

MoMO Project

console 898 ms

Result 12-35

:varchar 5

:varchar 6

Result 12-38

:varchar 7

:varchar 8

Result 12-41

Test

Status

2 All transactions reference valid senders

PASS

3 All transactions reference valid categories

PASS

4 All transaction tags reference valid transactions

PASS

4 rows

Database Consoles > MoMO Project > console 253:33 LF UTF-8 4 spaces Trial

Sample Transaction Query With JOINS(screenshot)

The screenshot shows a database console interface with a SQL query and its result set. The query is a complex JOIN query that retrieves transaction data along with sender and receiver user details and transaction categories.

```
SELECT 'SAMPLE TRANSACTION DATA WITH RELATIONSHIPS' AS '';
SELECT '-----' AS '';
SELECT
    t.transaction_id AS Txn_ID,
    s.full_name AS Sender,
    IFNULL(r.full_name, 'EXTERNAL') AS Receiver,
    tc.category_name AS Category,
    CONCAT('RWF ', FORMAT(t.amount, 2)) AS Amount,
    t.transaction_status AS Status,
    DATE_FORMAT(t.created_at, 'XY-%m-%d %H:%i') AS Created_At
FROM transactions t
JOIN users s 1.n<->1: ON t.sender_id = s.user_id
LEFT JOIN users r 1.n<->0.1: ON t.receiver_id = r.user_id
JOIN transaction_categories tc 1.n<->1: ON t.category_id = tc.category_id
ORDER BY t.transaction_id;
```

The result set displays 8 rows of transaction data:

	Sender	Receiver	Category	Amount	Status	Created_At
1	1 Arsene Hirwa	Vanessa Rusengwa	SEND_MONEY	RWF 6,000.00	SUCCESS	2025-01-10 14:30
2	2 Vanessa Rusengwa	Christa Ishimwe	CASH_OUT	RWF 30,000.00	SUCCESS	2024-12-20 10:20
3	3 Christa Ishimwe	EXTERNAL	PAY_BILL	RWF 10,000.00	FAILED	2025-01-13 08:20
4	4 Ank Micha	Axel Iragaba	SEND_MONEY	RWF 85,000.00	PENDING	2025-01-20 12:00
5	5 Axel Iragaba	Arsene Hirwa	AIRTIME_TOPUP	RWF 1,000.00	SUCCESS	2025-01-22 09:00
6	6 Abigael Kagabo	Ank Micha	MERCHANT_PAYMENT	RWF 40,000.00	SUCCESS	2025-01-18 10:00
7	7 Arsene Hirwa	Christa Ishimwe	MERCHANT_PAYMENT	RWF 20,000.00	FAILED	2025-01-19 11:00
8	8 Vanessa Rusengwa	EXTERNAL	CASH_OUT	RWF 60,000.00	REVERSED	2025-01-20 15:20

Database Setup Verification(screenshot)

The screenshot shows a database console interface with a SQL query for database setup verification. The query uses UNION ALL to check the existence and row counts of various tables in the database.

```
SELECT 'DATABASE SETUP VERIFICATION' AS '';
SELECT '-----' AS '';
SELECT 'Component' AS 'Table Name', 'Records' AS 'Row Count' FROM DUAL
UNION ALL
SELECT 'Table Name' 'Users', 'Row Count' CAST(COUNT(*) AS CHAR) FROM users
UNION ALL
SELECT 'Table Name' 'Transactions', 'Row Count' CAST(COUNT(*) AS CHAR) FROM transactions
UNION ALL
SELECT 'Table Name' 'Transaction Categories', 'Row Count' CAST(COUNT(*) AS CHAR) FROM transaction_categories
UNION ALL
SELECT 'Table Name' 'Tags', 'Row Count' CAST(COUNT(*) AS CHAR) FROM tags
UNION ALL
SELECT 'Table Name' 'Transaction Tags', 'Row Count' CAST(COUNT(*) AS CHAR) FROM transaction_tags
UNION ALL
SELECT 'Table Name' 'System Logs', 'Row Count' CAST(COUNT(*) AS CHAR) FROM system_logs;
```

The result set displays 7 rows of verification data:

Table Name	Row Count
Component	Records
Users	6
Transactions	8
Transaction Categories	7
Tags	7
Transaction Tags	11
System Logs	16

Security And Constraints Validation(screenshot)

The screenshot displays a database console interface with a dark theme. On the left, the 'Database Explorer' shows a project named 'MoMO Project' containing a database 'momo_sms_db' with tables like 'system_logs', 'tags', 'transaction_ca', 'transaction_tag', 'transactions', 'users', and 'Server Objects'. The main console area shows a SQL script for security and constraints validation. The script includes comments and SQL statements to check phone number format, ID number format, unique national ID numbers, and positive transaction amounts. The results are displayed in a table at the bottom right.

```
193 SELECT 'SECURITY CONSTRAINT VALIDATION' AS '';
194 SELECT '===== ' AS '';
195 SELECT 'Constraint Rule' AS 'Rule', 'Test Result' AS 'Status' FROM DUAL
196 UNION ALL
197 SELECT (Rule 'Phone Number Format (+250XXXXXXXXX)',
198        CASE WHEN COUNT(*) = 0 THEN 'PASS' ELSE 'FAIL' END
199        FROM users WHERE phone_number NOT REGEXP '^\\+250[0-9]{9}$'
200 UNION ALL
201 SELECT (Rule 'ID Number Format (16 digits)',
202        CASE WHEN COUNT(*) = 0 THEN 'PASS' ELSE 'FAIL' END
203        FROM users WHERE id_number NOT REGEXP '[0-9]{16}$'
204 UNION ALL
205 SELECT (Rule 'Unique National ID Numbers',
206        CASE WHEN COUNT(DISTINCT id_number) = COUNT(*) THEN 'PASS' ELSE 'FAIL' END
207        FROM users
208 UNION ALL
209 SELECT (Rule 'Positive Transaction Amounts',
210        CASE WHEN COUNT(*) = 0 THEN 'PASS' ELSE 'FAIL' END
211        FROM transactions WHERE amount <= 0;
```

Rule	Status
1 Constraint Rule	Test Result
2 Phone Number Format (+250XXXXXXXXX)	PASS
3 ID Number Format (16 digits)	PASS
4 Unique National ID Numbers	PASS
5 Positive Transaction Amounts	PASS

Conclusion

The Momo SMS database efficiently manages Mobile money transactions by organizing users, transactions, categories, tags, and system logs into a secure and relational structure. With enforced constraints, foreign keys, and indexes, it ensures data integrity, traceability, and reliable reporting. This design models real-world Mobile money operations accurately while remaining scalable, maintainable, and easy to analyze.