

# **DATABASE DOCUMENTATION**

## **1. Introduction**

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

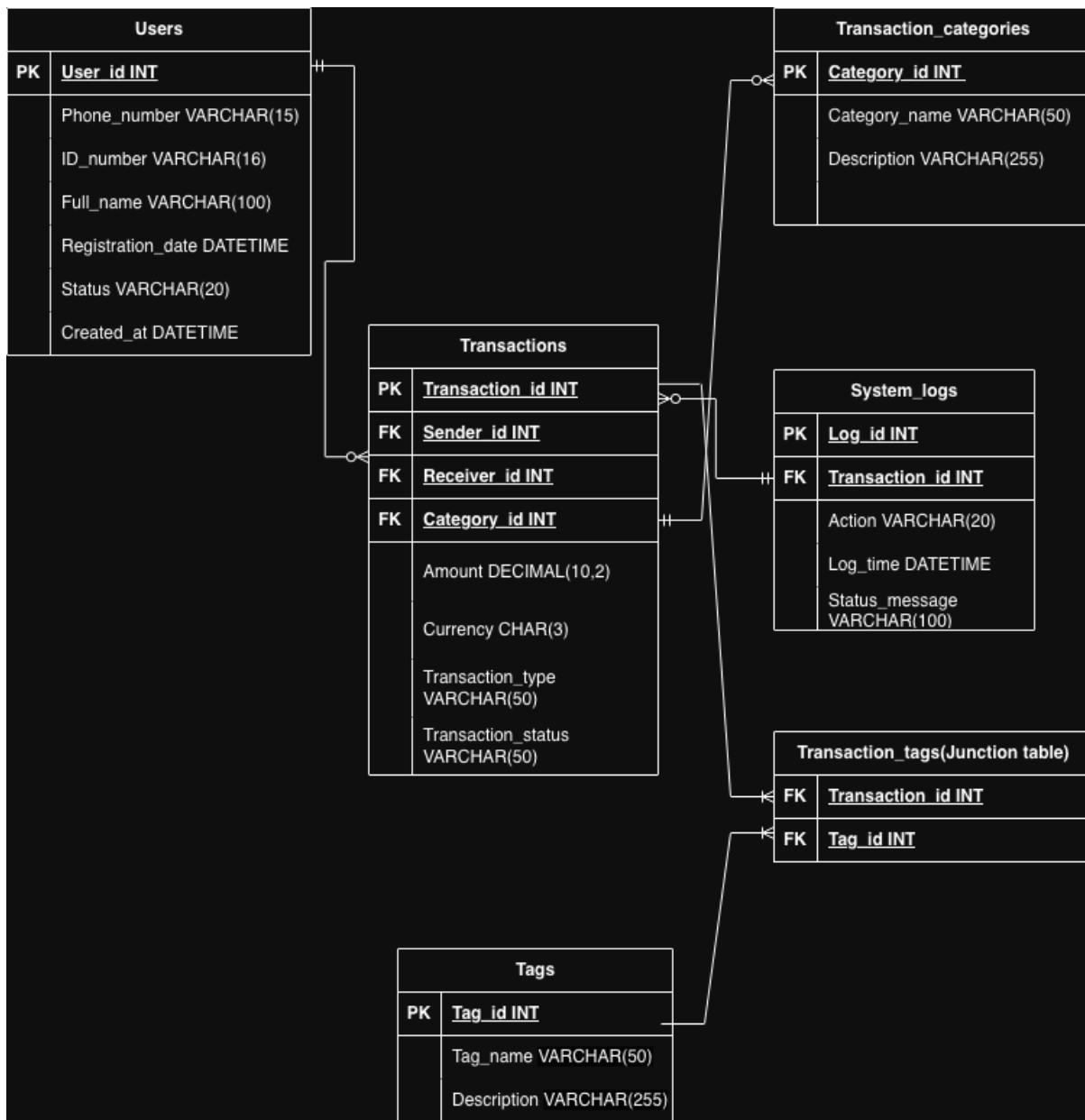
The database is designed to store user information, transaction records, transaction categories, tags, and system logs while ensuring data accuracy, security, and integrity.

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	Date user registered
status	VARCHAR(20)	User account status
created_at	DATETIME	Record creation time

#### **4.2 Transactions Table**

<b>Column name</b>	<b>Data type</b>	<b>Description</b>
tranaction_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**

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

#### **4.4 Tags Table**

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

#### **4.5 Transaction Logs Table**

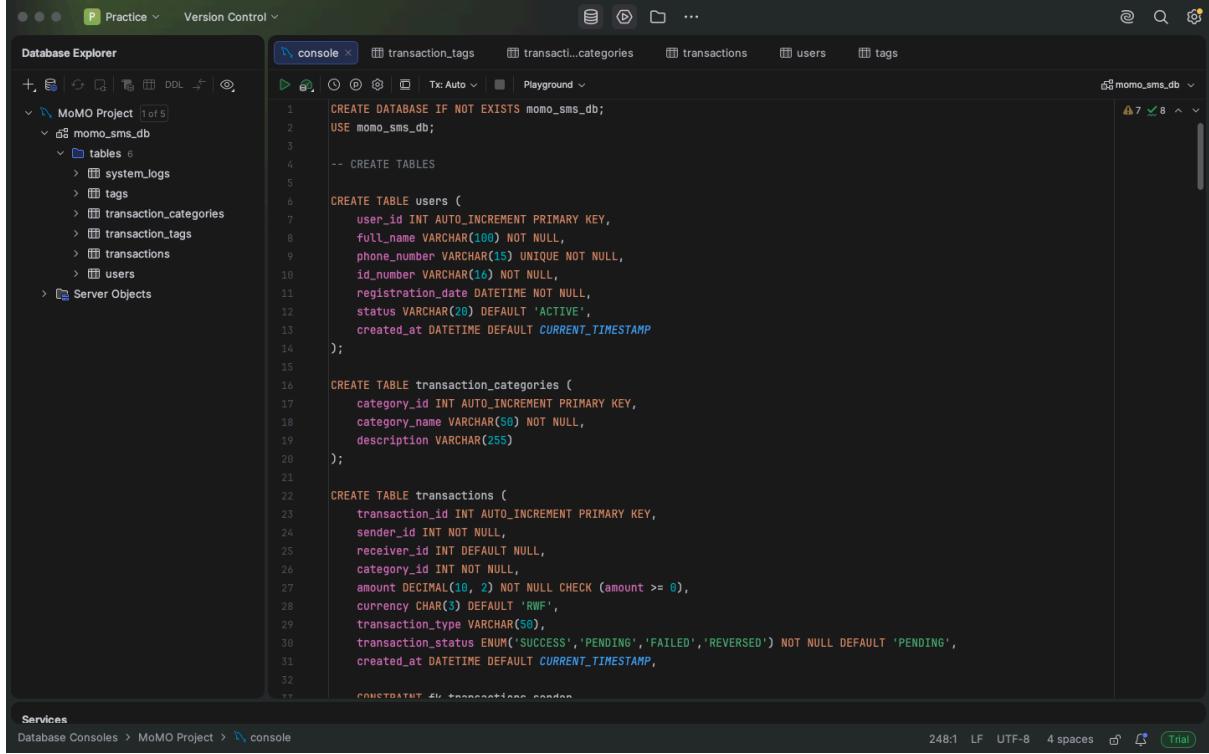
<b>Column name</b>	<b>Data type</b>	<b>Description</b>
transaction_id	INT	Linked transaction
tag_id	INT	Linked Tag

#### **4.6 System Logs Table**

<b>Column name</b>	<b>Data type</b>	<b>Description</b>
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:



```
CREATE DATABASE IF NOT EXISTS momo_sms_db;
USE momo_sms_db;

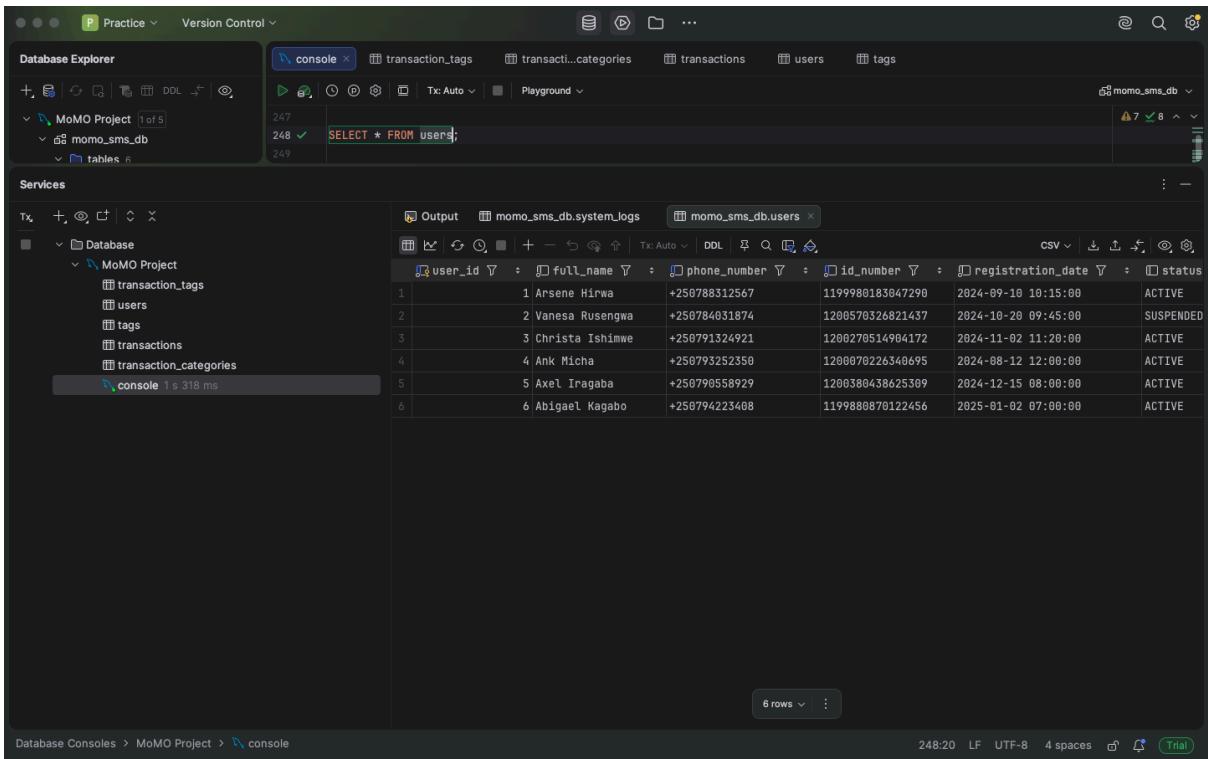
-- CREATE TABLES

CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    full_name VARCHAR(100) NOT NULL,
    phone_number VARCHAR(15) UNIQUE NOT NULL,
    id_number VARCHAR(16) NOT NULL,
    registration_date DATETIME NOT NULL,
    status VARCHAR(20) DEFAULT 'ACTIVE',
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE transaction_categories (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    category_name VARCHAR(50) NOT NULL,
    description VARCHAR(255)
);

CREATE TABLE transactions (
    transaction_id INT AUTO_INCREMENT PRIMARY KEY,
    sender_id INT NOT NULL,
    receiver_id INT DEFAULT NULL,
    category_id INT NOT NULL,
    amount DECIMAL(10, 2) NOT NULL CHECK (amount >= 0),
    currency CHAR(3) DEFAULT 'RWF',
    transaction_type VARCHAR(50),
    transaction_status ENUM('SUCCESS', 'PENDING', 'FAILED', 'REVERSED') NOT NULL DEFAULT 'PENDING',
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME
);
```

### Select:



```
SELECT * FROM users;
```

	user_id	full_name	phone_number	id_number	registration_date	status
1	1	Arsene Hirwa	+250788312567	1199980183047290	2024-09-10 10:15:00	ACTIVE
2	2	Vanessa Rusengwa	+250784031874	1200576326821437	2024-10-28 09:45:00	SUSPENDED
3	3	Christa Ishimwe	+250791324921	1200276514904172	2024-11-02 11:20:00	ACTIVE
4	4	Ank Micha	+250793252350	1200670226340695	2024-08-12 12:00:00	ACTIVE
5	5	Axel Iragaba	+250790558929	1200380438625309	2024-12-15 08:00:00	ACTIVE
6	6	Abigael Kagabo	+250794223408	1199886870122456	2025-01-02 07:00:00	ACTIVE

## Update:

The screenshot shows a database development environment with a dark theme. On the left, the Database Explorer pane displays a project named 'MoMO Project' with a database named 'momo\_sms\_db'. This database contains tables for 'transaction\_tags', 'tags', 'transaction\_categories', and 'transactions'. The 'users' table is also listed. On the right, the main workspace has two tabs: 'console' and 'momo\_sms\_db.users'. The 'console' tab contains the following SQL code:

```
249
250 UPDATE users
251 SET status = 'INACTIVE'
252 WHERE user_id = 2;
253 ?
254 ✓ SELECT * FROM users;
```

The 'momo\_sms\_db.users' tab shows a table with the following data:

_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	INACTIVE
3	Christa Ishimwe	+250791324921	1200270514904172	2024-11-02 11:20:00	ACTIVE
4	Ank Micha	+250793252359	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

At the bottom of the interface, there are status indicators: 'Database Consoles > MoMO Project > console', '254:21 LF UTF-8 4 spaces', and a green 'Trial' button.

**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 console interface with the following details:

- Database Explorer:** Shows the MoMo Project with the momo\_sms\_db database selected. Inside, there are tables: system\_logs, tags, transaction\_categories, transaction\_tags, transactions, and users.
- Console Tab:** Contains the following SQL code:

```
256 UPDATE users
257 SET status = 'INACTIVE'
258 WHERE user_id = 2;
259
260 SELECT * FROM users;
261
262 ① DELETE FROM users
263 WHERE full_name = 'Ank Michal';
264 SELECT * FROM users;
```
- Message Bar:** Displays the error message: [23000][1451] 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)
- Services Tab:** Shows a history of database operations with timestamps and command details.
- Bottom Status:** Shows the current session details: Database Consoles > MoMo Project > console, 264:21 LF UTF-8 4 spaces, and a Trial button.

## 5. Sample Queries Demonstrating Functionality

### Query One: View all successful transactions

The screenshot shows the DataGrip IDE interface. The Database Explorer on the left lists the MoMO Project and its tables: transaction\_tags, transaction\_categories, transactions, users, and tags. The Services panel shows a list of databases and tables. The main console window contains the following SQL query:

```
UNION ALL
SELECT 'Test All transaction tags reference valid transactions',
CASE WHEN NOT EXISTS (
    SELECT 1 FROM transaction_tags tt
    LEFT JOIN transactions t 1..n->1: ON tt.transaction_id = t.transaction_id
    WHERE t.transaction_id IS NULL
) THEN 'PASS' ELSE 'FAIL' END;

SELECT transaction_id, amount, transaction_status
FROM transactions
WHERE transaction_status = 'SUCCESS';
```

The output window shows the results of the query:

transaction_id	amount	transaction_status
1	6000.00	SUCCESS
2	30000.00	SUCCESS
5	1000.00	SUCCESS
6	40000.00	SUCCESS

Console status: 2 s 234 ms

### Query Two: View transactions sent by a specific user

The screenshot shows the DataGrip IDE interface. The Database Explorer on the left lists the MoMO Project and its tables: transaction\_tags, transaction\_categories, transactions, users, and tags. The Services panel shows a list of databases and tables. The main console window contains the following SQL query:

```
UNION ALL
SELECT 'Test All transaction tags reference valid transactions',
CASE WHEN NOT EXISTS (
    SELECT 1 FROM transaction_tags tt
    LEFT JOIN transactions t 1..n->1: ON tt.transaction_id = t.transaction_id
    WHERE t.transaction_id IS NULL
) THEN 'PASS' ELSE 'FAIL' END;

SELECT transaction_id, amount, transaction_status
FROM transactions
WHERE transaction_status = 'SUCCESS';

SELECT t.transaction_id, t.amount, u.full_name
FROM transactions t
JOIN users u 1..n->1: ON t.sender_id = u.user_id
WHERE u.full_name = 'Arsene Hirwa';
```

The output window shows the results of the query:

transaction_id	amount	full_name
1	6000.00	Arsene Hirwa
7	20000.00	Arsene Hirwa

Console status: 550 ms

## Query Three: View Transaction Logs

The screenshot shows a database console interface with the following details:

- Database Explorer:** Shows a project named "MoMO Project" with a database "momo\_sms\_db" containing tables: transaction\_tags, transaction\_categories, transactions, users, and system\_logs.
- Console:** Displays the SQL query:

```
SELECT transaction_id, action, log_time
FROM system_logs
ORDER BY log_time DESC;
```
- Output:** Shows the results of the query in a table format. The table has columns: transaction\_id, action, and log\_time. The data consists of 16 rows:

transaction_id	action	log_time
1	5 COMPLETE	2025-01-22 09:01:00
2	5 REQUEST	2025-01-22 09:08:05
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

## Foreign Key Integrity Tests (screenshot)

The screenshot shows a database development environment with the following interface elements:

- Database Explorer**: Shows a project named "MoMO Project" with a database named "momo\_sms\_db". Inside the database, there are tables: system\_logs, tags, transaction\_cat, transaction\_tags, transactions, and users.
- Services**: Shows a transaction named "console" with a duration of 898 ms.
- Console Tab**: Contains the following SQL code:

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

The results of the execution are displayed in a table:

Test	Status
All transactions reference valid senders	PASS
All transactions reference valid categories	PASS
All transaction tags reference valid transactions	PASS

At the bottom of the interface, there are status indicators: 253:33, LF, UTF-8, 4 spaces, and a Trial button.

## Sample Transaction Query With JOINs(screenshot)

The screenshot shows a database console interface with the following details:

**Console Tab:**

```
214 ✓ SELECT 'SAMPLE TRANSACTION DATA WITH RELATIONSHIPS' AS '';
215 ✓ SELECT '=====';
216 ✓ SELECT
217     t.transaction_id AS Txn_ID,
218     s.full_name AS Sender,
219     IFNULL(r.full_name, 'EXTERNAL') AS Receiver,
220     tc.category_name AS Category,
221     CONCAT('RWF ', FORMAT(t.amount, 2)) AS Amount,
222     t.transaction_status AS Status,
223     DATE_FORMAT(t.created_at, '%Y-%m-%d %H:%i') AS Created_At
224 FROM transactions t
225 JOIN users s 1..n->1: ON t.sender_id = s.user_id
226 LEFT JOIN users r 1..n->0..1: ON t.receiver_id = r.user_id
227 JOIN transaction_categories tc 1..n->1: ON t.category_id = tc.category_id
228 ORDER BY t.transaction_id;
```

**Services Tab:**

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

Database Consoles > MoMO Project > console

## Database Setup Verification(screenshot)

The screenshot shows a database console interface with the following details:

**Console Tab:**

```
176 ✓ SELECT 'DATABASE SETUP VERIFICATION' AS '';
177 ✓ SELECT =====;
178 ✓ SELECT 'Component' AS 'Table Name', 'Records' AS 'Row Count' FROM DUAL
179 UNION ALL
180 SELECT Table Name 'Users', Row Count CAST(COUNT(*) AS CHAR) FROM users
181 UNION ALL
182 SELECT Table Name 'Transactions', Row Count CAST(COUNT(*) AS CHAR) FROM transactions
183 UNION ALL
184 SELECT Table Name 'Transaction Categories', Row Count CAST(COUNT(*) AS CHAR) FROM transaction_categories
185 UNION ALL
186 SELECT Table Name 'Tags', Row Count CAST(COUNT(*) AS CHAR) FROM tags
187 UNION ALL
188 SELECT Table Name 'Transaction Tags', Row Count CAST(COUNT(*) AS CHAR) FROM transaction_tags
189 UNION ALL
190 SELECT Table Name 'System Logs', Row Count CAST(COUNT(*) AS CHAR) FROM system_logs;
```

**Services Tab:**

Component	Records
Users	6
Transactions	8
Transaction Categories	7
Tags	7
Transaction Tags	11
System Logs	16

Database Consoles > MoMO Project > console

## Security And Constraints Validation(screenshot)

The screenshot shows a database management interface with a 'Database Explorer' sidebar on the left and a 'console' tab in the center. The 'Database Explorer' shows a project named 'MoMO Project' with a database 'momo\_sms\_db' containing tables like 'system\_logs', 'tags', 'transaction\_ca', 'transaction\_ta', 'transactions', and 'users'. The 'console' tab displays SQL code for validating security constraints across these tables. The results are shown in a table below:

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

At the bottom, the status bar indicates '193:44 LF UTF-8 4 spaces'.

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

## 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.