MoMo SMS Database Design Documentation

1. ERD Analysis and Entity Design

Based on our analysis of the MoMo XML data structure, we identified four main entities that handle the mobile money transaction processing system:

Entities:

- **Users** - handles customer data for people sending/receiving money
- **Categories** - stores different types of transactions like payments, transfers etc.
- **Transactions** - main table with all the transaction records
- **System_Logs** - keeps track of what happens when data gets processed

Entity Details:

Users Table:

- user_id(Primary Key): unique ID for each customer
- name: stores customer names

Categories Table:

- category_id: unique ID for transaction types
- category_name: name of transaction type (like "Debit", "Credit")

Transactions Table:

- transaction_id(Primary Key): unique transaction ID
- amount: how much money in the transaction
- fee: processing fees charged
- new_balance: account balance after transaction
- Transaction_date: when transaction happened
- Category: description of transaction type
- category_id(Foreign Key): links to categories table
- user_id(Foreign Key): links to users table

System_Logs Table:

- log_id(Primary Key) : unique log ID
- log_timestamp: when log was created
- transaction_id(Foreign Key) : links to specific transaction

Relationships Between Tables:

- Users to Transactions = 1:M (one user can have many transactions)
- Categories to Transactions = 1:M (one category applies to many transactions)
- Transactions to System_Logs = 1:1 (each transaction creates one log entry)Users to categories = M:M (many categories can be applied to many users)
- Users to Categories through a bridge table = M:N (Each of the entities above are linked to the bridge table in a 1:M relationship making it a M:N relationship)

Primary Keys

- User_id
- Category_id
- Transaction_id
- log_id

Foreign Keys:

- transactions.category_id connects to categories.category_id
- transactions.user_id connects to users.user_id
- system_logs.transaction_id connects to transactions.transaction_id

2. Design Decision Explanation

Our team decided to normalize the database by putting users and categories in separate tables instead of repeating this information everywhere. This reduces data duplication and makes the database more organized.

We made transactions the center of our design since that's the most important data we are handling. All other tables connect to it through foreign keys, which ensures data integrity - you can't have a transaction without a valid user and category.

We decided to use our own unique ID numbers for transaction_id, category_id, and user_id. This decision was influenced by our findings that ID numbers present in the XML file were not constant. Some had a transaction ID number, while others had a Txt ID number or both of them. So this inconsistency saw us using auto-incremental values that SQL generated.

For money values, we chose DECIMAL(15,2) because regular numbers aren't precise enough for financial calculations. We learned this can cause rounding errors with money, so DECIMAL gives us exact precision. DATETIME was picked for timestamps since we need to track exactly when transactions happen for auditing purposes.

We added indexes on columns that will be queried often like transaction_date, user_id, and category_id. This should make queries faster when we're looking up user transaction history or generating reports by category.

The System_Logs table was added because we need to track what the system is doing for troubleshooting and monitoring. Having a 1:1 relationship with transactions means we can trace every transaction that gets processed.

We kept both category and category_id in the transactions table even though it seems redundant. This lets us get category info quickly without always having to join tables, which improves performance for basic queries.

The design uses standard MySQL data types and should scale well as the system grows while keeping all the ACID properties needed for financial data.

3. Data Dictionary

USERS TABLE			
Column name	Data type	Constraints	Description
user	VARCHAR(25)	None	Customer's full name
user_id	INT(auto_increment)	NOT NULL, PRIMARY KEY	Auto-generated user ID number
CATEGORIES TABLE			
category_id	INT(auto_increment)	PRIMARY KEY, NOT NULL	Unique ID for each category
category_name	VARCHAR(50)	None	Type of transaction (Debit, Credit, etc..)
TRANSACTIONS TABLE			

transaction_id	INT(auto_increment)	PRIMARY KEY, NOT NULL	Unique ID for each transaction
amount	DECIMAL(15,2)	None	Amount of money with decimal presicion
fee	INT	None	Processing fee charged per transaction
new-balance	DECIMAL(15,2)	None	Balance after the fee is charged
transacton_date	DATETIME	None	When the transaction occurred
category	VARCHAR(50)	None	Description of the transaction type from the categories table
category_id	INT	FOREIGN KEY, NOT NULL	Links to the category table
user_id	INT	FOREIGN KEY, NOT NULL	Links to the users table
SYSTEM LOGS TABLE			
log_id	INT(auto_increment)	PRIMARY KEY, NOT NULL	Unique ID number for each log entry auto-generated
log_timestamp	DATETIME	NoneNone	hen the log entry was created
transaction_id	INT	FOREIGN KEY, NOT NULL	Links to the related transaction

4. Sample Database Queries here

Sample DML statements to insert test data

USERS Table

```
INSERT INTO users (name) VALUES
('Ayomide'),
('Neza'),
('Rowan'),
('Duke'),
('Habeeb');
```

Description: Adds 5 sample users to the database.

CATEGORIES Table

```
INSERT INTO categories (category_name) VALUES
('Debit'),
('Credit'),
('Electricity'),
('Airtime'),
('Food');
```

Description: Creates 5 transaction categories (Debit, Credit, Electricity, Airtime, Food).

TRANSACTIONS Table

```
INSERT INTO transactions (amount, fee, new_balance, transaction_date, category,
category_id, user_id) VALUES
(1000.00, 100, 5000.00, '2025-09-18 09:00:00', 'Debit', 1, 1),
(2000.00, 100, 2950.00, '2025-09-18 10:00:00', 'Credit', 2, 2),
(500.00, 50, 1500.00, '2025-09-18 11:00:00', 'Electricity', 3, 3),
(1200.00, 100, 3800.00, '2025-09-18 12:00:00', 'Airtime', 4, 4),
(750.00, 50, 4250.00, '2025-09-18 13:00:00', 'Food', 5, 5);
```

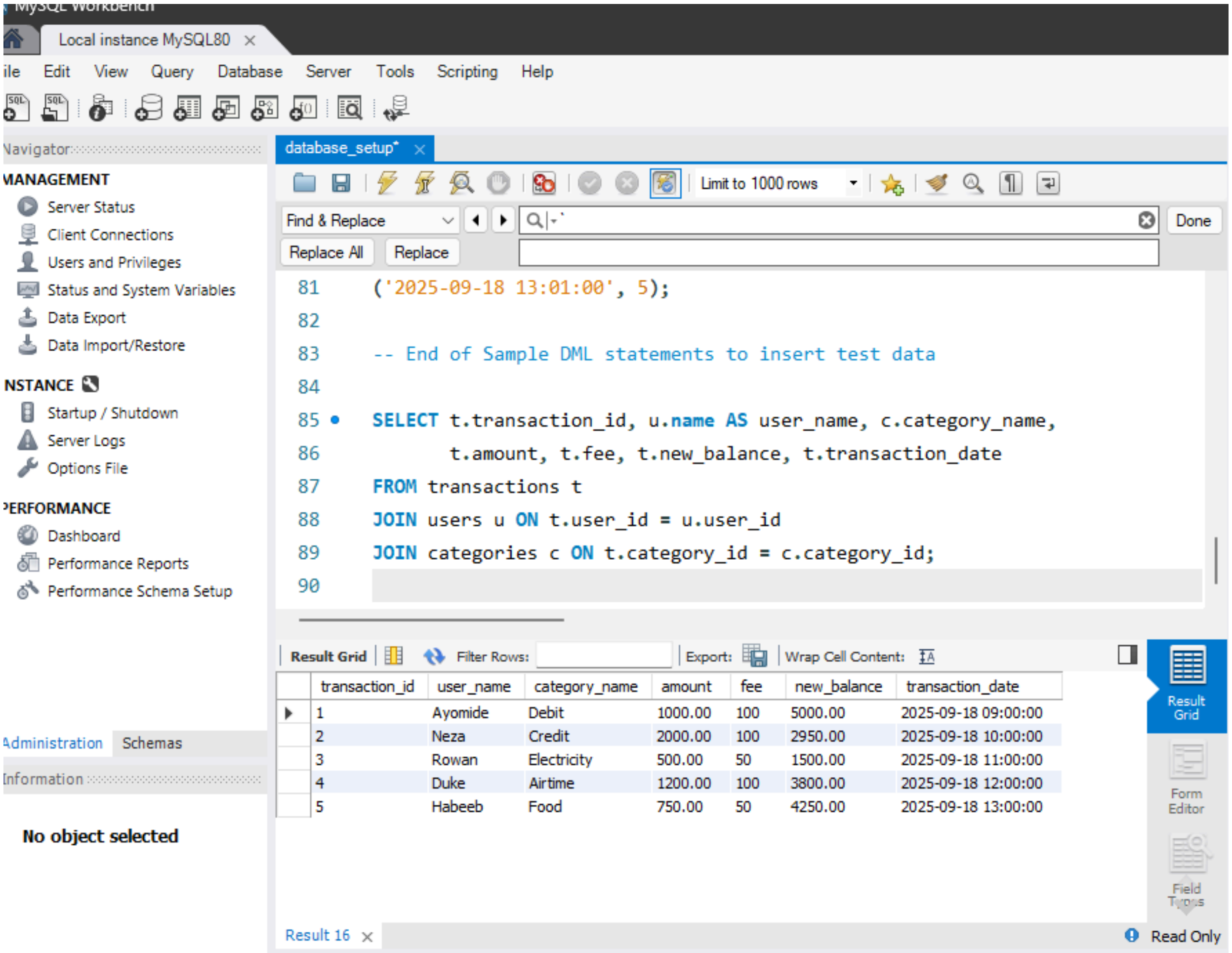
Description: Inserts 5 sample transactions with amounts, fees, balances, and timestamps. Each transaction is linked to a user and category.

SYSTEM_LOGS Table

```
INSERT INTO System_Logs (log_timestamp, transaction_id) VALUES
('2025-09-18 09:01:00', 1),
('2025-09-18 10:01:00', 2),
('2025-09-18 11:01:00', 3),
('2025-09-18 12:01:00', 4),
('2025-09-18 13:01:00', 5);
```

List all transactions with user and category

```
SELECT t.transaction_id, u.name AS user_name, c.category_name,
       t.amount, t.fee, t.new_balance, t.transaction_date
FROM transactions t
JOIN users u ON t.user_id = u.user_id
JOIN categories c ON t.category_id = c.category_id;
```

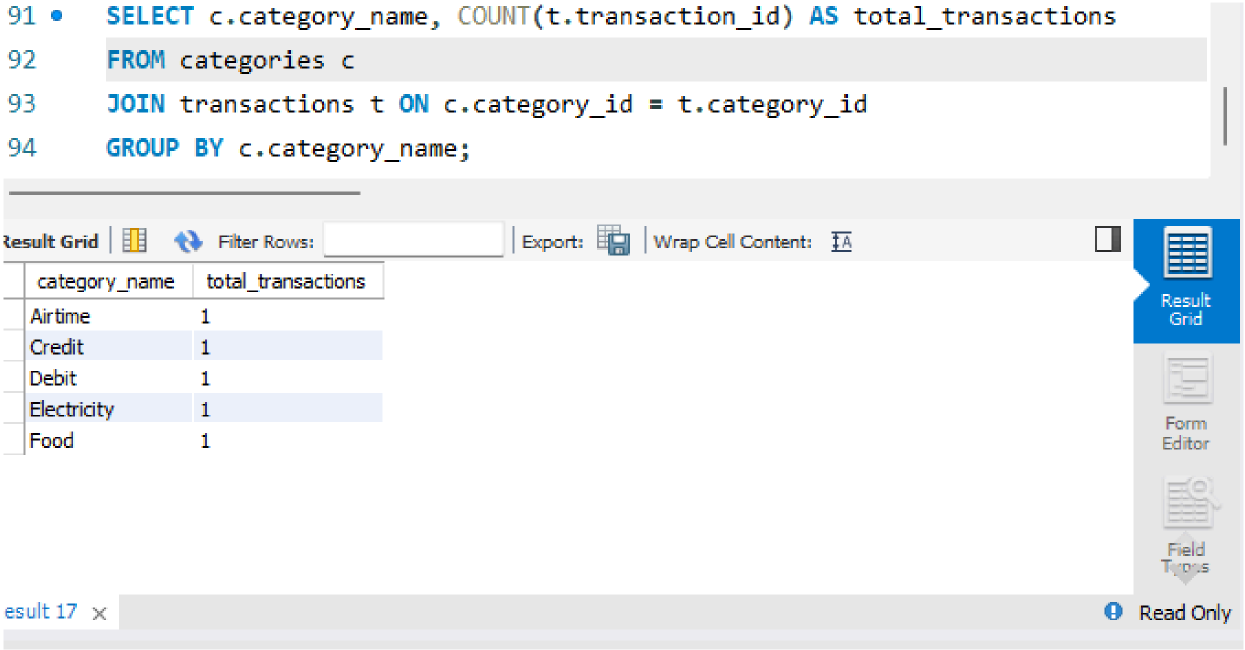


```
('2025-09-18 13:01:00', 5);

-- End of Sample DML statements to insert test data

SELECT t.transaction_id, u.name AS user_name, c.category_name,
       t.amount, t.fee, t.new_balance, t.transaction_date
FROM transactions t
JOIN users u ON t.user_id = u.user_id
JOIN categories c ON t.category_id = c.category_id;
```

transaction_id	user_name	category_name	amount	fee	new_balance	transaction_date
1	Ayomide	Debit	1000.00	100	5000.00	2025-09-18 09:00:00
2	Neza	Credit	2000.00	100	2950.00	2025-09-18 10:00:00
3	Rowan	Electricity	500.00	50	1500.00	2025-09-18 11:00:00
4	Duke	Airtime	1200.00	100	3800.00	2025-09-18 12:00:00
5	Habeeb	Food	750.00	50	4250.00	2025-09-18 13:00:00

Count number of transactions per category

SELECT c.category_name, COUNT(t.transaction_id) AS total_transactions
FROM categories c
JOIN transactions t ON c.category_id = t.category_id
GROUP BY c.category_name;



```sql
SELECT c.category_name, COUNT(t.transaction_id) AS total_transactions
FROM categories c
JOIN transactions t ON c.category_id = t.category_id
GROUP BY c.category_name;
```

category_name	total_transactions
Airtime	1
Credit	1
Debit	1
Electricity	1
Food	1

5. Security and Data Accuracy Rules

Foreign Key Constraints: We decide to put foreign key constraints to prevent bad data from getting into the database. For example, you can't create a transaction with a category_id that doesn't exist in the categories table.

Data Type Constraints:

- DECIMAL(15,2) ensures money calculations are accurate
- AUTO_INCREMENT prevents duplicate IDs
- Primary keys make sure each record is unique

Performance Indexes: We added indexes on commonly searched columns instead of all the columns to make queries run faster.

Testing Constraint Violations: When we tried to insert invalid data, the database correctly rejected it:

This fails because category_id 999 doesn't exist
INSERT INTO transactions (amount, fee, category_id, user_id)
VALUES (1000.00, 50, 999, 1);
Error: Cannot add or update a child row: foreign key constraint fails.

Prevent negative transaction amounts

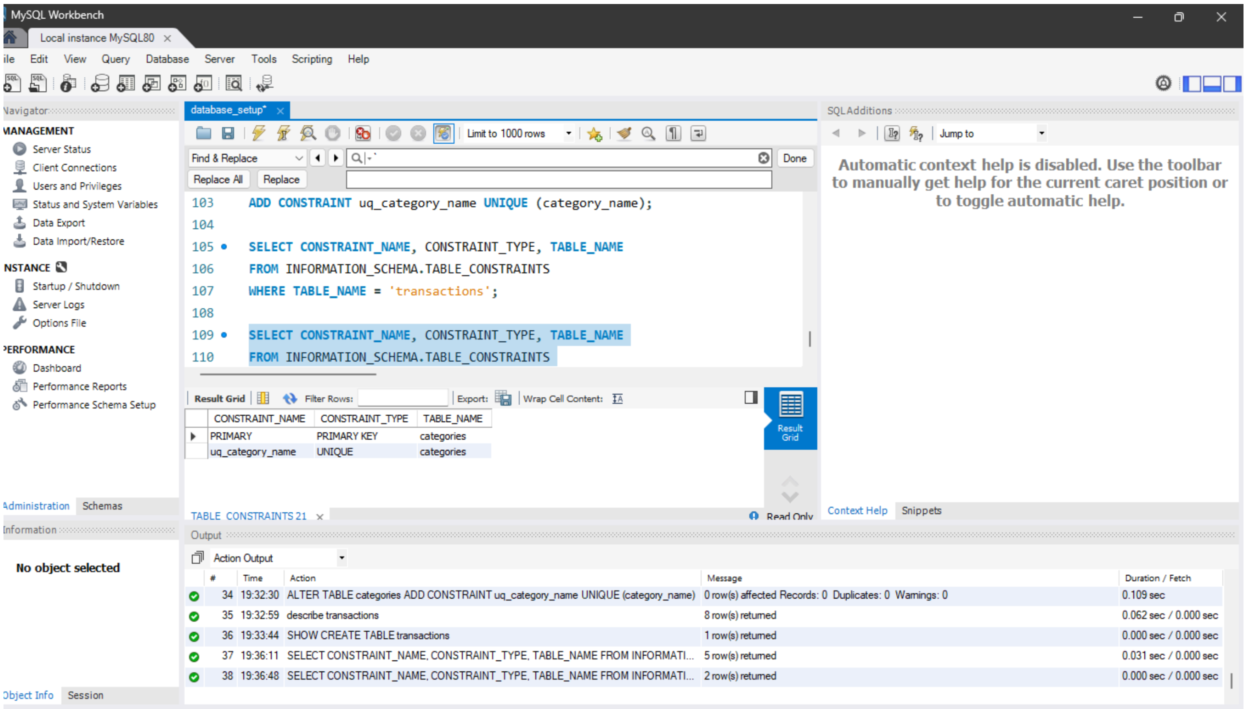
```
ALTER TABLE transactions
ADD CONSTRAINT chk_amount_positive CHECK (amount > 0);
```

Prevent negative balances

```
ALTER TABLE transactions
ADD CONSTRAINT chk_balance_nonnegative CHECK (new_balance >= 0);
```

Ensure category names are unique

```
ALTER TABLE categories
ADD CONSTRAINT uq_category_name UNIQUE (category_name);
```

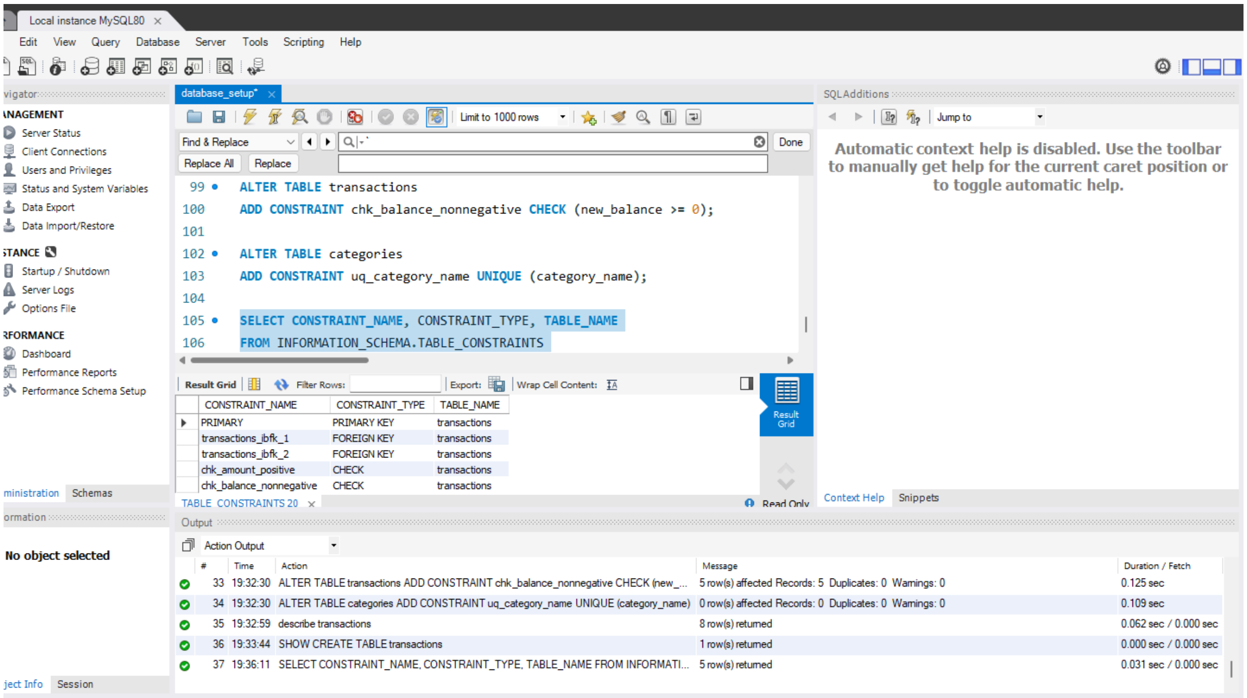


```
ADD CONSTRAINT uq_category_name UNIQUE (category_name);

SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE TABLE_NAME = 'transactions';

SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
```

CONSTRAINT_NAME	CONSTRAINT_TYPE	TABLE_NAME
PRIMARY	PRIMARY KEY	categories
uq_category_name	UNIQUE	categories

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

	#	Time	Action	Message	Duration / Fetch
✓	34	19:32:30	ALTER TABLE categories ADD CONSTRAINT uq_category_name UNIQUE (category_name)	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.109 sec
✓	35	19:32:59	describe transactions	8 row(s) returned	0.062 sec / 0.000 sec
✓	36	19:33:44	SHOW CREATE TABLE transactions	1 row(s) returned	0.000 sec / 0.000 sec
✓	37	19:36:11	SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME FROM INFORMATI...	5 row(s) returned	0.031 sec / 0.000 sec
✓	38	19:36:48	SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME FROM INFORMATI...	2 row(s) returned	0.000 sec / 0.000 sec



```sql
ALTER TABLE transactions
ADD CONSTRAINT chk_balance_nonnegative CHECK (new_balance >= 0);

ALTER TABLE categories
ADD CONSTRAINT uq_category_name UNIQUE (category_name);

SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
```

CONSTRAINT_NAME	CONSTRAINT_TYPE	TABLE_NAME
PRIMARY	PRIMARY KEY	transactions
transactions_ibfk_1	FOREIGN KEY	transactions
transactions_ibfk_2	FOREIGN KEY	transactions
chk_amount_positive	CHECK	transactions
chk_balance_nonnegative	CHECK	transactions

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

```
#   Time      Action                                                                        Message                                                      Duration / Fetch
33  19:32:30  ALTER TABLE transactions ADD CONSTRAINT chk_balance_nonnegative CHECK (new_...  5 row(s) affected Records: 5  Duplicates: 0  Warnings: 0  0.125 sec
34  19:32:30  ALTER TABLE categories ADD CONSTRAINT uq_category_name UNIQUE (category_name)  0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0  0.109 sec
35  19:32:59  describe transactions                                                         8 row(s) returned                                            0.062 sec / 0.000 sec
36  19:33:44  SHOW CREATE TABLE transactions                                                1 row(s) returned                                            0.000 sec / 0.000 sec
37  19:36:11  SELECT CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME FROM INFORMATI...         5 row(s) returned                                            0.031 sec / 0.000 sec
```

6. SQL → JSON Mapping Documentation

- **Users**

users table → user.schema.json
SQL column user_id → user_id
SQL column name → name

- **Categories**

categories table → category.schema.json
SQL column category_id → category_id
SQL column category_name → category_name

- **Transactions**

transactions table → transaction.schema.json

SQL columns map directly:
- transaction_id → transaction_id
- amount → amount
- fee → fee
- new_balance → new_balance
- transaction_date → transaction_date

- category → category
- category_id → category_id
- user_id → user_id

- **System Logs**

System_Logs table → system_log.schema.json
SQL columns map directly:
- log_id → log_id
- log_timestamp → log_timestamp
- transaction_id → transaction_id

- **Transaction Full (Nested): Joins data from transactions, users, categories, and system_logs into a nested JSON object**

transaction_full.schema.json combines multiple entities:
- transactions → core transaction object
- users → nested sender/receiver
- categories → nested category_details
- system_logs → nested logs array

P09