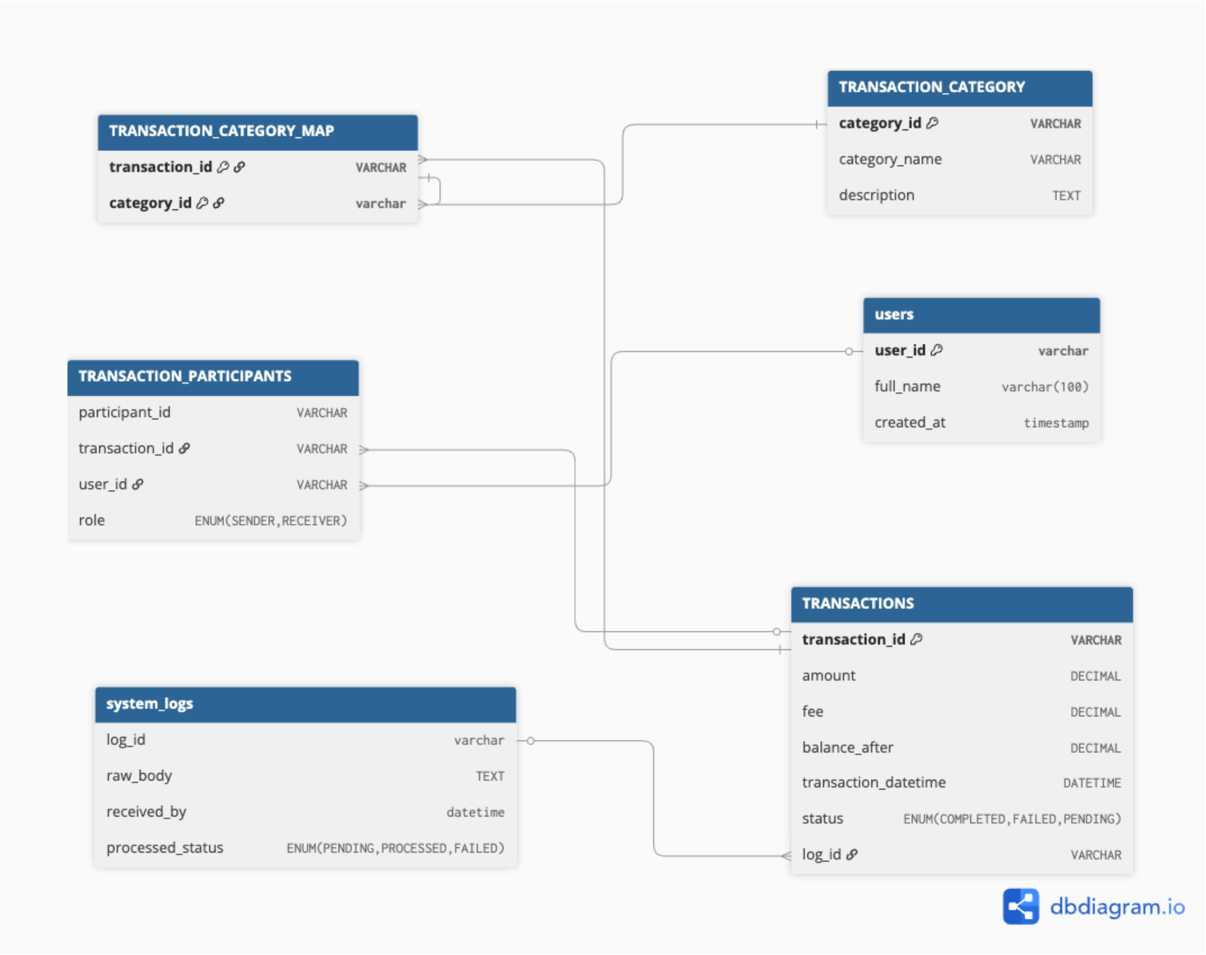
Database Design Document

Entity Relation diagram (ERD)



Design rationale and justification

- Why we separated the raw logs and transaction data

The design maintains a clear separation between raw SMS data (SYSTEM_LOGS)
and processed transaction data (TRANSACTIONS). This architectural decision
provides several benefits:

1. Complete audit trail - raw messages are never modified
2. Reprocessing capability - failed transactions can be reprocessed from
   original SMS

3. Data integrity - structured transaction data is validated separately
4. System reliability - processing failures don't lose original data

- The 1:1 relationship between SYSTEM_LOGS and TRANSACTIONS ensures that each

  SMS can only generate one transaction, preventing duplicate processing while
  maintaining the ability to trace any transaction back to its source message.

- MANY-TO-MANY CATEGORY RELATIONSHIP

  The M:N relationship between TRANSACTIONS and TRANSACTION_CATEGORY through the TRANSACTION_CATEGORY_MAP junction table allows flexible categorization.

  BENEFITS:
  - A single transaction can belong to multiple categories (e.g., a bank deposit that also involves a user payment)
  - Sophisticated reporting and analysis without data duplication
  - Easy addition of new categories without schema changes
  - Flexible business logic for transaction classification

  The junction table uses a composite primary key (transaction_id, category_id)
  to prevent duplicate category assignments while maintaining referential integrity through foreign keys

- PARTICIPANT TRACKING DESIGN

  The TRANSACTION_PARTICIPANTS table implements a role-based model where each
  transaction records who was involved and their role (SENDER/RECEIVER).

  This design supports:
  1. Complete transaction history per user
  2. Bilateral transaction tracking (sender and receiver perspectives)
  3. Future extensibility for multi-party transactions

4. Clear accountability and reporting capabilities
5. Separation of concerns (users vs. their participation in transactions)

The participant_id primary key allows the same user to have multiple roles
in different transactions while maintaining unique participant records.

DATA TYPE AND CONSTRAINT CHOICES

DECIMAL(12,2) for Financial Amounts:
- Prevents floating-point errors in monetary calculations
- Supports amounts up to 999,999,999,999.99 RWF
- Exact precision for accounting requirements

VARCHAR(20) for IDs:
- Human-readable identifiers from SMS messages
- Maintains uniqueness while being memorable
- Sufficient length for alphanumeric transaction IDs

ENUM Types for Status Fields:
- Ensures data consistency (only valid values accepted)
- Reduces storage compared to VARCHAR
- Self-documenting (valid states are explicit)
- Database-level validation

CHECK Constraints:
- Enforces business rules at database level
- Prevents invalid data entry (negative amounts, etc.)
- Cannot be bypassed by application code
- Provides automatic validation

Data dictionary

USERS Table

Column	Data_type	Constraints	Description
user_id	varchar(20)	Primary Key	Unique identifier
full_name	varchar(100)	NOT NULL	User full name
created_at	TIMESTAMP	Default current	Account creation

		timestamp	timestamp

SYSTEM_LOGS Table

Column	**Data_type**	**Constraints**	**Description**
log_id	varchar(20)	Primary Key	Unique Identifier
raw_body	TEXT	NOT NULL	Complete un processed sms
received_at	DATETIME	NOT NULL	Time stamp when the sms was received
processed_status	ENUM	DEFAULT PENDING	Processed status : PENDING , PROCESSED,FAILED

TRANSACTIONS Table

Column	**Data_type**	**Constraints**	**Description**
transaction_id	varchar(20)	Primary Key	Unique identifier
amount	decimal(12,2)	NOT NULL < check > 0	Transaction amount in RWF
fee	decimal(10,2)	DEFAULT 0 CHECK >=0	Fee charged for transaction
balance_after	decimal(12,2)		Account balance after transaction
transaction_datetime	Datetime	NOT NULL	When transaction occured
status	ENUM		Transaction status : COMPLETED ,FAILED ,

			PENDING
log_id	varchar(20)	Foreign Key	Foreign key to sms log

TRANSACTION_CATEGORY Table

Column	Data_type	Constraints	Description
category_id	varchar(20)	Primary Key	Unique identifier
category_name	varchar(50)	Unique not null	Display name of the category
description	TEXT	null	description

TRANSACTION_CATEGORY_MAP Table

Column	Data_type	Constraints	Description
transaction_id	varchar(20)	Primary key , Foreign key	Foraging key to the transaction table
category_id	varchar(20)	Primary key , Foreign key	Foreign key to the transaction_category table

TRANSACTION_PARTICIPANTS

Column	Data_type	Constraints	Description
participant_id	varchar(20)	Primary key	Unique identifier
transaction_id	varchar(20)	Foreign key	Foreign key to the transaction table
user_id	varchar(20)	Foreign key	Foregin key to

			the users table
role	ENUM	NOT NULL	Participant role : receiver , sender

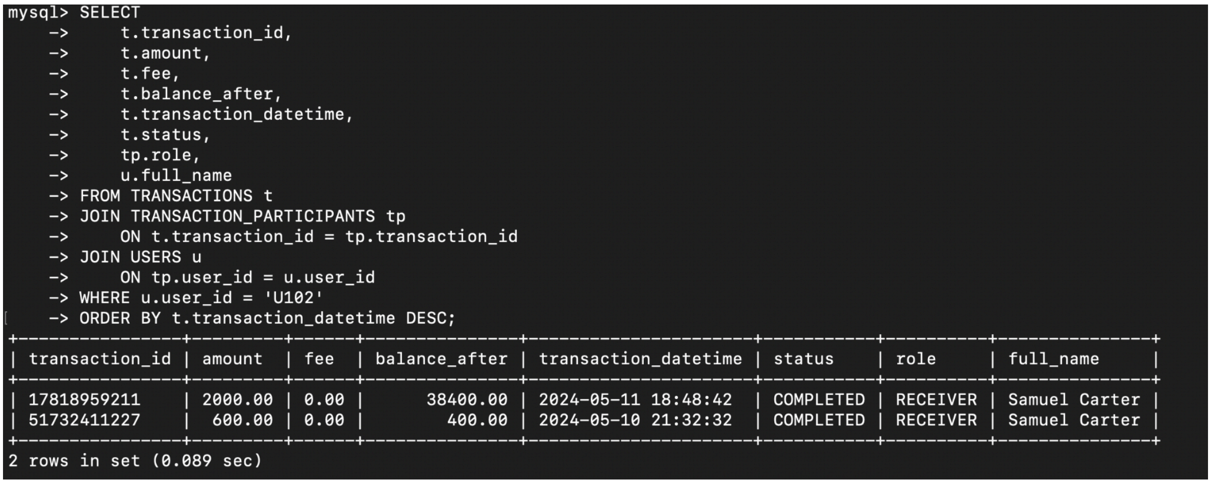
Sample queries

TRANSACTION HISTORY FOR A SPECIFIC USER

Query

```
SELECT
    t.transaction_id,
    t.amount,
    t.fee,
    t.balance_after,
    t.transaction_datetime,
    t.status,
    tp.role,
    u.full_name
FROM TRANSACTIONS t
JOIN TRANSACTION_PARTICIPANTS tp
    ON t.transaction_id = tp.transaction_id
JOIN USERS u
    ON tp.user_id = u.user_id
WHERE u.user_id = 'U102'
ORDER BY t.transaction_datetime DESC;
```

Result



```
mysql> SELECT
    ->     t.transaction_id,
    ->     t.amount,
    ->     t.fee,
    ->     t.balance_after,
    ->     t.transaction_datetime,
    ->     t.status,
    ->     tp.role,
    ->     u.full_name
    -> FROM TRANSACTIONS t
    -> JOIN TRANSACTION_PARTICIPANTS tp
    ->     ON t.transaction_id = tp.transaction_id
    -> JOIN USERS u
    ->     ON tp.user_id = u.user_id
    -> WHERE u.user_id = 'U102'
    -> ORDER BY t.transaction_datetime DESC;
+----------------+---------+------+---------------+----------------------+-----------+----------+---------------+
| transaction_id | amount  | fee  | balance_after | transaction_datetime | status    | role     | full_name     |
+----------------+---------+------+---------------+----------------------+-----------+----------+---------------+
| 17818959211    | 2000.00 | 0.00 |      38400.00 | 2024-05-11 18:48:42  | COMPLETED | RECEIVER | Samuel Carter |
| 51732411227    |  600.00 | 0.00 |        400.00 | 2024-05-10 21:32:32  | COMPLETED | RECEIVER | Samuel Carter |
+----------------+---------+------+---------------+----------------------+-----------+----------+---------------+
2 rows in set (0.089 sec)
```

Explanation

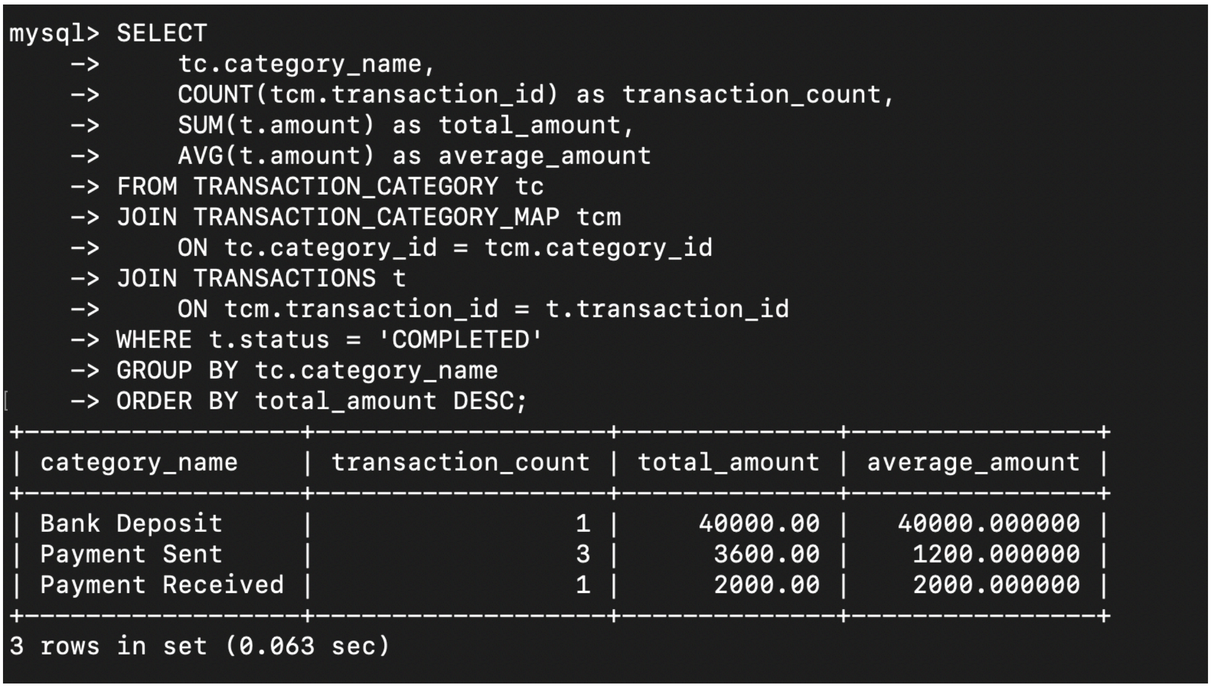
This query joins three tables to provide complete transaction context for user U102 (Samuel Carter), showing their role in each transaction.

TRANSACTIONS BY CATEGORY

Query

```
SELECT
    tc.category_name,
    COUNT(tcm.transaction_id) as transaction_count,
    SUM(t.amount) as total_amount,
    AVG(t.amount) as average_amount
FROM TRANSACTION_CATEGORY tc
JOIN TRANSACTION_CATEGORY_MAP tcm
    ON tc.category_id = tcm.category_id
JOIN TRANSACTIONS t
    ON tcm.transaction_id = t.transaction_id
WHERE t.status = 'COMPLETED'
GROUP BY tc.category_name
ORDER BY total_amount DESC;
```

Result



```
mysql> SELECT
    ->     tc.category_name,
    ->     COUNT(tcm.transaction_id) as transaction_count,
    ->     SUM(t.amount) as total_amount,
    ->     AVG(t.amount) as average_amount
    -> FROM TRANSACTION_CATEGORY tc
    -> JOIN TRANSACTION_CATEGORY_MAP tcm
    ->     ON tc.category_id = tcm.category_id
    -> JOIN TRANSACTIONS t
    ->     ON tcm.transaction_id = t.transaction_id
    -> WHERE t.status = 'COMPLETED'
    -> GROUP BY tc.category_name
    -> ORDER BY total_amount DESC;
+------------------+-------------------+--------------+----------------+
| category_name    | transaction_count | total_amount | average_amount |
+------------------+-------------------+--------------+----------------+
| Bank Deposit     |                 1 |     40000.00 |   40000.000000 |
| Payment Sent     |                 3 |      3600.00 |    1200.000000 |
| Payment Received |                 1 |      2000.00 |    2000.000000 |
+------------------+-------------------+--------------+----------------+
3 rows in set (0.063 sec)
```

Explanation

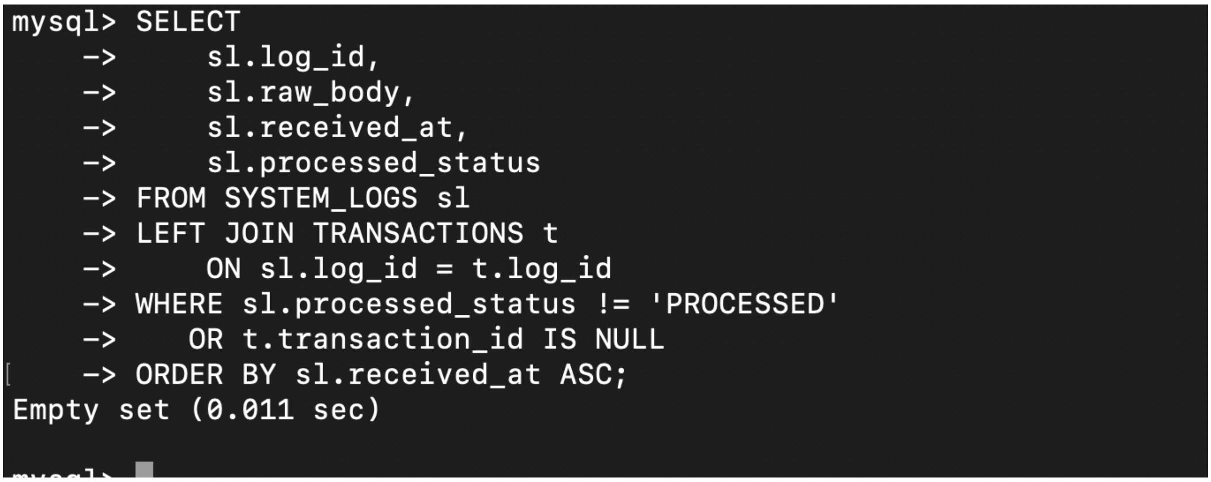
Provides financial summary by transaction category, useful for business intelligence and reporting.

UNPROCESSED SMS LOGS

Query

```
SELECT
    sl.log_id,
    sl.raw_body,
    sl.received_at,
    sl.processed_status
FROM SYSTEM_LOGS sl
LEFT JOIN TRANSACTIONS t
    ON sl.log_id = t.log_id
WHERE sl.processed_status != 'PROCESSED'
   OR t.transaction_id IS NULL
ORDER BY sl.received_at ASC;
```

Result



```
mysql> SELECT
    ->     sl.log_id,
    ->     sl.raw_body,
    ->     sl.received_at,
    ->     sl.processed_status
    -> FROM SYSTEM_LOGS sl
    -> LEFT JOIN TRANSACTIONS t
    ->     ON sl.log_id = t.log_id
    -> WHERE sl.processed_status != 'PROCESSED'
    ->    OR t.transaction_id IS NULL
    -> ORDER BY sl.received_at ASC;
Empty set (0.011 sec)

mysql>
```

Explanation

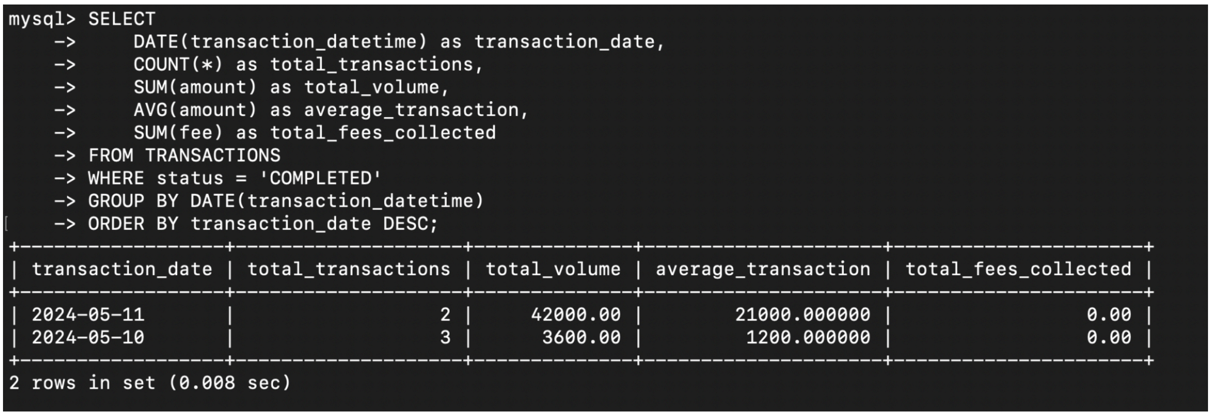
Critical for monitoring system health and identifying processing failures or backlogs. And for this database (No unprocessed logs in current dataset - all SMS messages have been successfully processed)

DAILY TRANSACTION SUMMARY

Query

```
SELECT
   DATE(transaction_datetime) as transaction_date,
   COUNT(*) as total_transactions,
   SUM(amount) as total_volume,
   AVG(amount) as average_transaction,
   SUM(fee) as total_fees_collected
FROM TRANSACTIONS
WHERE status = 'COMPLETED'
GROUP BY DATE(transaction_datetime)
ORDER BY transaction_date DESC;
```

Result



```
mysql> SELECT
    ->     DATE(transaction_datetime) as transaction_date,
    ->     COUNT(*) as total_transactions,
    ->     SUM(amount) as total_volume,
    ->     AVG(amount) as average_transaction,
    ->     SUM(fee) as total_fees_collected
    -> FROM TRANSACTIONS
    -> WHERE status = 'COMPLETED'
    -> GROUP BY DATE(transaction_datetime)
    -> ORDER BY transaction_date DESC;
+------------------+--------------------+--------------+---------------------+----------------------+
| transaction_date | total_transactions | total_volume | average_transaction | total_fees_collected |
+------------------+--------------------+--------------+---------------------+----------------------+
| 2024-05-11       |                  2 |     42000.00 |        21000.000000 |                 0.00 |
| 2024-05-10       |                  3 |      3600.00 |         1200.000000 |                 0.00 |
+------------------+--------------------+--------------+---------------------+----------------------+
2 rows in set (0.008 sec)
```

Explanation

Enables daily business reporting and trend identification for decision-making.

Security and data integrity rules

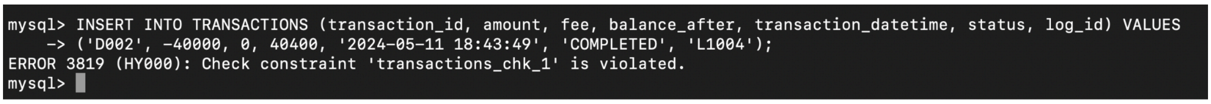
CHECK CONSTRAINTS

Check constraints enforce business rules at the database level, preventing invalid data entry regardless of application logic.

Query

INSERT INTO TRANSACTIONS (transaction_id, amount, fee, balance_after, transaction_datetime, status, log_id) VALUES
('D002', -40000, 0, 40400, '2024-05-11 18:43:49', 'COMPLETED', 'L1004');

Result



```
mysql> INSERT INTO TRANSACTIONS (transaction_id, amount, fee, balance_after, transaction_datetime, status, log_id) VALUES
    -> ('D002', -40000, 0, 40400, '2024-05-11 18:43:49', 'COMPLETED', 'L1004');
ERROR 3819 (HY000): Check constraint 'transactions_chk_1' is violated.
mysql> 
```

Explanation

As you can see above when we try to insert a transaction record with negative(-) amount the check constraints are violated the same thing happens when the amount is equal to 0 . this ensures that no invalid amount is ever in the table

UNIQUE CONSTRAINTS

Unique constraints prevent duplicate data and enforce one-to-one relationships.

Example : all the primary keys are unique and don’t allow duplicate values

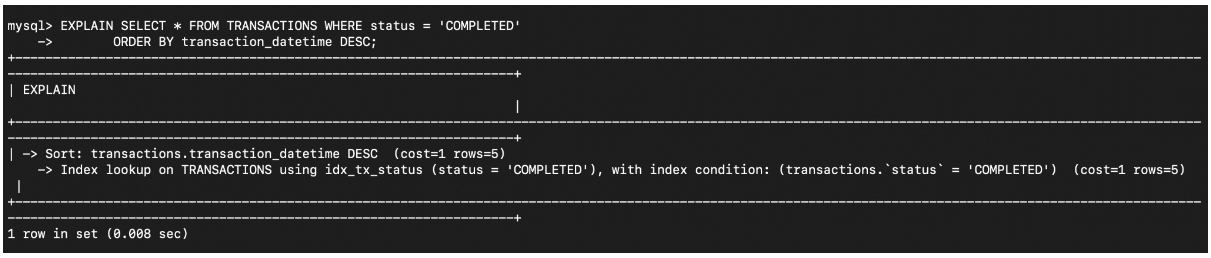
INDEXES

Indexes allow for fast searching of records in tables

Query

```
EXPLAIN SELECT * FROM TRANSACTIONS WHERE status = 'COMPLETED'
    ORDER BY transaction_datetime DESC;
```

Result



```
mysql> EXPLAIN SELECT * FROM TRANSACTIONS WHERE status = 'COMPLETED'
    ->        ORDER BY transaction_datetime DESC;
+--------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
| EXPLAIN
                                                                   |
+--------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
| -> Sort: transactions.transaction_datetime DESC  (cost=1 rows=5)
    -> Index lookup on TRANSACTIONS using idx_tx_status (status = 'COMPLETED'), with index condition: (transactions.`status` = 'COMPLETED')  (cost=1 rows=5)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------+
1 row in set (0.008 sec)
```

ADDITIONAL SECURITY MEASURES

- NOT NULL CONSTRAINTS: Critical fields (amounts, names, dates) cannot be null, preventing incomplete records that could compromise data integrity.

- DEFAULT VALUES: Automatic timestamp generation for created_at ensures proper audit trails without relying on application logic.

- DECIMAL PRECISION: Financial amounts use DECIMAL(12,2) to prevent rounding errors inherent in FLOAT types. This is critical for accounting accuracy and financial compliance.

- IMMUTABLE LOGS: SYSTEM_LOGS table design encourages append-only operations, preserving complete audit history. Raw SMS data is never modified or deleted, maintaining compliance with financial regulations.

- COMPOSITE KEYS: Junction table (TRANSACTION_CATEGORY_MAP) uses composite primary key (transaction_id, category_id) preventing duplicate mappings while maintaining referential integrity.

- VARCHAR LENGTH LIMITS: All VARCHAR columns have appropriate length limits to prevent buffer overflow attacks and ensure data quality.

- TIMESTAMP TRACKING: All major tables include timestamp fields (created_at, received_at, transaction_datetime) providing complete temporal audit trail.