

Banking Core - Business Flow Documentation

System Overview

Banking Core is a **production-grade core banking system** implementing true double-entry accounting with stored-procedure-driven money movement. The system uses:

- **Next.js 16** for frontend and API
- **MySQL 8** for data persistence
- **JWT** for authentication
- **BDT** as the sole currency

Design Philosophy

Correctness > Convenience > Speed

All financial operations are atomic, with no partial transactions possible. Money movement occurs exclusively through stored procedures with explicit row locking.

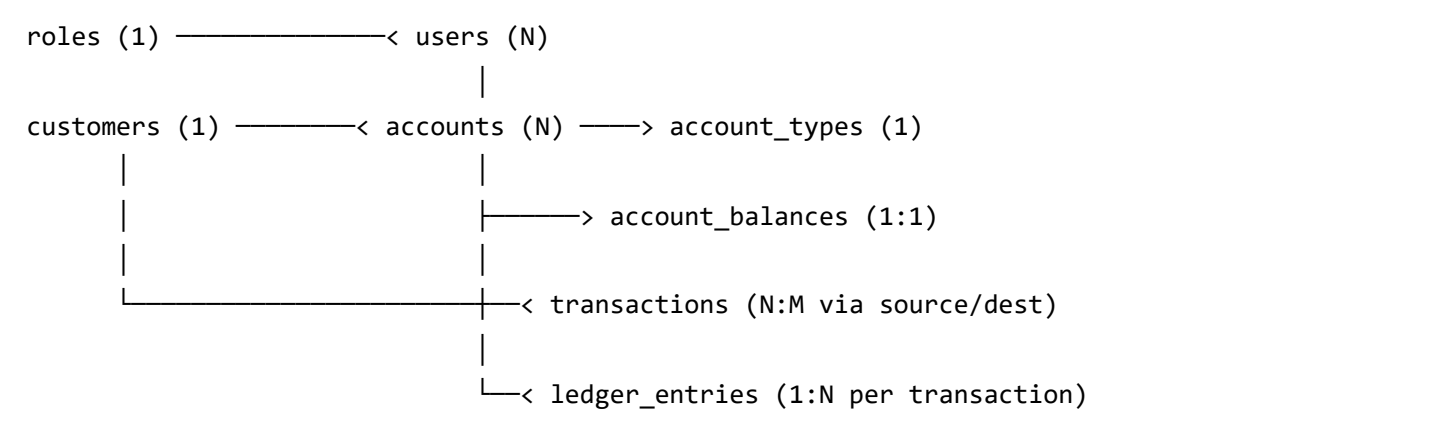
Database Schema Overview

Core Tables

Table	Purpose	Key Fields
roles	User role definitions	code, name, permissions (JSON)
users	Internal staff accounts	email, password_hash, role_id
customers	Bank customers	customer_number, kyc_status, status
accounts	Bank accounts	account_number, customer_id, account_type_id, status

Table	Purpose	Key Fields
account_types	Account classifications	code (SAVINGS, CHECKING, FIXED, INTERNAL)
account_balances	Current balance cache	available_balance, pending_balance, version
transactions	Transaction headers	transaction_reference, amount, status, source/dest account
ledger_entries	Double-entry records	entry_type (DEBIT/CREDIT), balance_after
transaction_audit	Audit trail	mirrors ledger with timestamp
audit_logs	System activity log	actor, action_type, entity, before/after state

Entity Relationship Diagram (Textual)



Role-Wise Business Flows

1. Customer Role

Login URL: `/login`

Authentication Flow

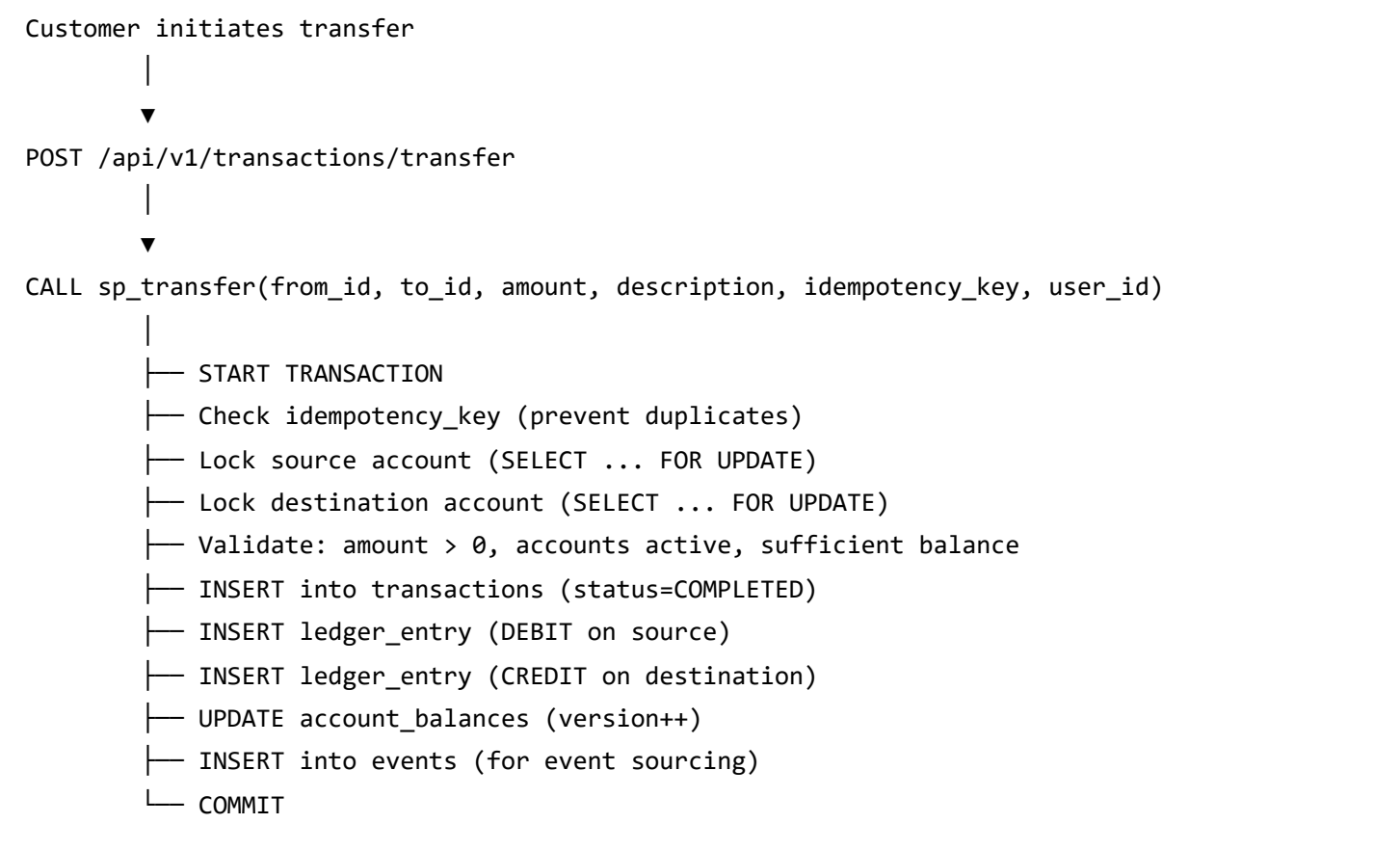
1. Customer enters email/password at `/login`

2. API POST /api/v1/auth/login validates credentials
3. JWT token generated with type: 'customer'
4. Customer redirected to /customer/dashboard

Operations

Operation	API Endpoint	DB Impact
View Accounts	GET /api/v1/accounts	SELECT from accounts + account_balances JOIN
View Transactions	GET /api/v1/accounts/[id]/statement	SELECT from transactions + ledger_entries
Transfer Funds	POST /api/v1/transactions/transfer	Calls sp_transfer stored procedure
Download Statement PDF	GET /api/v1/accounts/[id]/statement/pdf	SELECT + PDF generation
Change Password	POST /api/v1/customer/profile/password	UPDATE customers.password_hash

Transfer Flow (Detailed)



2. Banker/Teller Role

Login URL: /internal/login

Authentication Flow

- 1. Banker enters credentials at /internal/login
- 2. API validates user with role_id = 2 (BANKER)
- 3. JWT token with type: 'user' generated
- 4. Redirected to /banker/dashboard

Operations

Operation	API Endpoint	DB Impact
Onboard Customer	POST /api/v1/banker/customers/create	INSERT customers + accounts + account_balances

Operation	API Endpoint	DB Impact
Open Account	POST /api/v1/banker/accounts	INSERT accounts + account_balances
Cash Deposit	POST /api/v1/banker/deposits	Calls sp_deposit stored procedure
Cash Withdrawal	POST /api/v1/banker/withdrawals	Calls sp_withdraw stored procedure
View Customer Accounts	GET /api/v1/banker/accounts/[id]	SELECT accounts + balances
Approve KYC	PUT /api/v1/banker/customers/[id]	UPDATE customers.kyc_status
Search Ledger	GET /api/v1/banker/ledger	SELECT ledger_entries + accounts

Deposit Flow (sp_deposit)

Banker initiates deposit for customer account

|

▼

CALL sp_deposit(account_id, amount, description, banker_id)

|

├─ START TRANSACTION

├─ Lock BANK-CASH-001 (internal cash account)

├─ Lock customer account

├─ Validate: account active, amount > 0

├─ INSERT transaction (source=BANK-CASH, dest=customer)

├─ INSERT ledger_entry (DEBIT from BANK-CASH)

├─ INSERT ledger_entry (CREDIT to customer)

├─ UPDATE account_balances for both accounts

├─ INSERT outbox event (for async processing)

└─ COMMIT

Key Constraint

- All deposits come FROM BANK-CASH-001 (account_id=999)
- All withdrawals go TO BANK-CASH-001
- This ensures double-entry integrity

3. Auditor Role

Login URL: /internal/login

Authentication Flow

- 1. Auditor logs in with `role_id = 3` (AUDITOR)
- 2. Read-only access granted via permission `["read:*"]`
- 3. Redirected to `/auditor/dashboard`

Operations (ALL READ-ONLY)

Operation	API Endpoint	DB Impact
View All Transactions	GET /api/v1/auditor/transactions	SELECT transactions (system-wide)
View Ledger	GET /api/v1/auditor/ledger	SELECT ledger_entries (full)
View Audit Logs	GET /api/v1/auditor/audit-logs	SELECT audit_logs
View Customers	GET /api/v1/auditor/customers	SELECT customers
View Accounts	GET /api/v1/auditor/accounts	SELECT accounts + balances
Export Transactions PDF	GET /api/v1/auditor/export-pdf/transactions	SELECT + PDF generation
Export Ledger PDF	GET /api/v1/auditor/export-pdf/ledger	SELECT + PDF generation
Export Daily Totals PDF	GET /api/v1/auditor/export-pdf/daily-totals	SELECT aggregated data
Export Monthly Summary PDF	GET /api/v1/auditor/export-pdf/monthly-summary	SELECT aggregated data

Audit Trail Generation

Every significant action creates an audit log entry:

```
INSERT INTO audit_logs (  
    actor_id, actor_type, actor_role,  
    action_type, entity_type, entity_id,  
    before_state, after_state, metadata  
)
```

Actions logged: ACCOUNT_CREATED, ACCOUNT_FROZEN, USER_LOGIN, USER_LOGOUT, PASSWORD_CHANGED, PDF_EXPORTED, etc.

4. Admin Role

Login URL: /internal/login

Authentication Flow

- 1. Admin logs in with role_id = 1 (ADMIN)
- 2. Full access granted via permission ["*\"]
- 3. Redirected to /admin/dashboard

Operations

Operation	API Endpoint	DB Impact
Manage Users	GET/POST /api/v1/admin/users	SELECT/INSERT/UPDATE users
Manage Roles	GET/POST /api/v1/admin/roles	SELECT/INSERT/UPDATE roles
System Config	GET/PUT /api/v1/admin/config	SELECT/UPDATE system_config
Run EOD Jobs	POST /api/v1/admin/eod/run	Executes batch procedures
Post Interest	POST /api/v1/admin/interest/post	Batch ledger updates
Rebuild Balances	POST /api/v1/admin/reports/rebuild	Calls sp_rebuild_balance
View Jobs	GET /api/v1/admin/jobs	SELECT system_jobs

EOD (End-of-Day) Processing

Admin triggers EOD



- 1. Generate daily_account_totals for all active accounts
- 2. Calculate monthly_account_summaries if month-end
- 3. Post interest for eligible SAVINGS accounts
- 4. Generate top_accounts_monthly rankings
- 5. Update system_jobs with completion status

Transaction Management

ACID Compliance

Property	Implementation
Atomicity	All stored procedures use START TRANSACTION / COMMIT / ROLLBACK
Consistency	Triggers and CHECK constraints prevent invalid states
Isolation	SELECT ... FOR UPDATE ensures row-level locking
Durability	InnoDB engine with write-ahead logging

Idempotency

Each transfer has an idempotency_key checked before processing:

```
SELECT response_body FROM idempotency_keys
WHERE idempotency_key = ? AND expires_at > NOW()
```

If found, cached response is returned. Otherwise, new transaction proceeds.

Concurrency Control

```
-- Lock source account
SELECT available_balance FROM account_balances
WHERE account_id = ? FOR UPDATE;

-- Lock destination account
SELECT available_balance FROM account_balances
WHERE account_id = ? FOR UPDATE;
```

Locks are released only on COMMIT/ROLLBACK.

Data Integrity & Constraints

Financial Constraints

Constraint	Enforcement
Non-negative balance	sp_transfer checks v_from_balance >= p_amount
Currency lock	All amounts in BDT (hardcoded)
Account status	Operations only on status = 'ACTIVE' accounts
Append-only ledger	No UPDATE/DELETE on ledger_entries

Referential Integrity

Parent Table	Child Table	Relationship
customers	accounts	customer_id → id
accounts	account_balances	account_id → id
accounts	ledger_entries	account_id → id
transactions	ledger_entries	transaction_id → id
roles	users	role_id → id

Precision

All monetary values stored as `DECIMAL(18,4)` to avoid floating-point errors.

Reporting & Audit Flow

Transaction Reports

1. User requests statement for account
2. Query joins `transactions` , `ledger_entries` , `accounts`
3. Results formatted and returned (JSON or PDF)

Audit Reports

1. Auditor queries `/auditor/audit-logs`
2. Filters by `action_type`, `entity_type`, date range
3. Export to PDF includes watermark "Banking Core – Audit Copy"
4. Export action itself logged to `audit_logs` (PDF_EXPORTED)

Daily/Monthly Summaries

Pre-calculated during EOD processing:

- `daily_account_totals` : opening_balance, credits, debits, closing_balance per account per day
- `monthly_account_summaries` : aggregated monthly data
- `top_accounts_monthly` : rankings by balance, volume, transaction count

Error Handling & Rollback Logic

Stored Procedure Error Handler

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
    GET DIAGNOSTICS CONDITION 1
        @sqlstate = RETURNED_SQLSTATE,
        @errno = MYSQL_ERRNO,
        @text = MESSAGE_TEXT;
    ROLLBACK;
    SET p_status = 'FAILED';
    SET p_message = CONCAT('Error: ', COALESCE(@text, 'Unknown'));
END;
```

Validation Error Response

Before any DB write, validations occur:

- Amount > 0
- Account exists and active
- Sufficient balance
- Same account check (no self-transfer)

If validation fails, ROLLBACK is called before any changes.

API Error Responses

HTTP Code	Meaning
200	Success
400	Validation error
401	Unauthorized (no/invalid token)
403	Forbidden (insufficient permissions)
404	Resource not found
500	Server error (logged, no sensitive data exposed)

Summary Tables

CRUD Operations by Role

Role	Create	Read	Update	Delete
Customer	Transfers only	Own accounts/transactions	Password only	None
Banker	Customers, Accounts, Deposits, Withdrawals	Assigned customers	KYC status	None
Auditor	None	All (read-only)	None	None
Admin	Users, Roles, Config	All	All (except ledger)	Users only

Stored Procedures Summary

Procedure	Called By	DB Tables Affected
sp_transfer	Customer, Banker	transactions, ledger_entries, account_balances, idempotency_keys, events, outbox
sp_deposit	Banker	transactions, ledger_entries, account_balances, events, outbox
sp_withdraw	Banker	transactions, ledger_entries, account_balances, events, outbox
sp_rebuild_balance	Admin	account_balances (recalculated from ledger_entries)

Conclusion

Banking Core demonstrates a well-structured core banking architecture with:

- 1. **Clear separation of concerns** via role-based access control
- 2. **Financial integrity** through stored procedures with explicit locking

3. **Audit compliance** via comprehensive logging of all actions
4. **Scalability patterns** including idempotency, event sourcing, and outbox pattern

This documentation serves as a comprehensive reference for understanding the system's business logic and database interactions.