PORTSAID INTERNATIONAL BANK

TEAM MEMBERS:
1-YOUSSEF IHAB 23-101138
2-YOUSSEF ASHOUSH 23-101049
3-OMAR ABOELNAGA 23-101149
4-HAMDY ELSAEED 23-101232

Deliverable #1 Template – Database Project (Fall 2025)

1. English Requirements (Business Rules)

System Overview

Portsaid International Bank web app for retail banking customers and bank employees.

Two user roles: Customers (self-service banking) and Employees (ops/teller/loan staff).

Business Scenario

Customers manage their accounts: view balances/history, move money, request loans, and see overdraft events.

Employees operate on behalf of customers: create customer profiles, perform cash operations, review/update loans, adjust account status, run summaries/reports, and clean old records.

Functional Requirements

Authentication with username/PIN; role detected (customer vs employee).

Customer capabilities:

View owned accounts and balances.

View transaction history with filters (date range, type).

Deposit and withdraw cash on owned accounts.

Transfer funds between accounts (own → other).

Request loans and view loan status.

View overdraft events recorded on their accounts.

Employee capabilities:

Create customer profiles (username, PIN, full name, email, phone, address, national ID).

Perform deposits/withdrawals for a specified account.

Review all loans and update loan status.

Update account status (ACTIVE/FROZEN/CLOSED).

Run account summary reports (inflow/outflow, overdraft counts).

Delete pending loans;

purge overdraft events older than N days.

Data persistence:

Accounts, Customers, Employees, Transactions, Transfers, Loans, OverDraftEvents stored in SQL Server.

All state-changing actions logged to Transactions; transfers mirrored for both accounts.

Overdraft attempts logged to OverDraftEvents.

Business Rules and Constraints

Role-based access: customer UIs are limited to their own accounts; employee UIs expose operational tools.

Account status must be ACTIVE to transact (deposits/withdrawals/transfers) or request loans.

Transfers validate source and destination accounts exist and are ACTIVE.

Withdrawals/transfers with insufficient funds are blocked and create an overdraft event.

Loan requests allowed on ACTIVE accounts; loan status changes only via employee actions.

Pending loans can be deleted; non-pending loans cannot.

Overdraft event cleanup deletes only events older than the chosen age.

Transactions are always recorded with timestamps; transfers generate paired in/out entries.

Customer creation requires unique username and national ID.

Employees can change account status but not customer status.

Report summaries use deterministic aggregates (total inflow, total outflow, overdraft count) per account.

Illustrative Constraints (examples)

A transfer cannot proceed if source balance < transfer amount; an overdraft event is recorded and the transfer is rejected.

A deposit/withdrawal is rejected if the account status is not ACTIVE.

A loan can only be requested on an ACTIVE account; its initial status is PENDING until an employee updates it.

A pending loan is the only loan type eligible for deletion; approved/rejected/closed loans are retained.

Overdraft events older than the specified “days” threshold are eligible for bulk deletion; newer events remain.

Usernames and national IDs must be unique when creating customers; duplicates are rejected

2. Entity–Relationship Diagram (ERD)

Create the ERD using any CASE tool that uses course-accepted notation. Your ERD must include: - All entities - All attributes - Primary keys clearly marked - All relationships - Correct cardinality & participation constraints - Weak/strong entities (if any)

> **Attach or paste the ERD below, or submit as a separate file.**

---

3. Relational Schema

```
CUSTOMERS(
   customer_id PK,
   username UQ,
   pin,
   name,
   email,
   phone,
   address,
   national_id UQ,
   status
)

EMPLOYEES(
   employee_id PK,
   username UQ,
   pin,
   name,
   email,
   phone,
   role,
   status
)

ACCOUNTS(
   account_number PK,
   customer_id FK → CUSTOMERS(customer_id),
   account_type,
   balance,
   currency,
   status,
   date_opened
)

TRANSACTIONS(
   transaction_id PK,
   account_number FK → ACCOUNTS(account_number),
   transaction_type,
   amount,
   timestamp,
   performed_by,
   note,
   balance_after,
   reference_code
)

TRANSFERS(
   transfer_id PK,
   from_account FK → ACCOUNTS(account_number),
   to_account FK → ACCOUNTS(account_number),
   amount,
   timestamp,
   status,
   note
)

LOANS(
   loan_id PK,
   account_number FK → ACCOUNTS(account_number),
   principal,
   balance_remaining,
   rate,
   term_months,
   start_date,
   status,
   next_due_date
)

OVERDRAFTEVENTS(
   event_id PK,
   account_number FK → ACCOUNTS(account_number),
   amount,
   occurred_at,
   note,
   balance_after
)
```

RELATIONSHIPS (from FKs; cardinality/participation)

1. CUSTOMERS 1 —— N ACCOUNTS
   - ACCOUNTS.customer_id is NOT NULL ⇒ each Account must belong to exactly 1 Customer
   - a Customer can have 0..N Accounts
2. ACCOUNTS 1 —— N TRANSACTIONS
   - TRANSACTIONS.account_number is NOT NULL ⇒ each Transaction belongs to exactly 1 Account
   - an Account can have 0..N Transactions
3. ACCOUNTS 1 —— N LOANS
   - LOANS.account_number is NOT NULL ⇒ each Loan belongs to exactly 1 Account
   - an Account can have 0..N Loans
4. ACCOUNTS 1 —— N OVERDRAFTEVENTS
   - OVERDRAFTEVENTS.account_number is NOT NULL ⇒ each OverDraftEvent belongs to exactly 1 Account
   - an Account can have 0..N OverDraftEvents
5. ACCOUNTS 1 —— N TRANSFERS (Sender role)
   - TRANSFERS.from_account is NOT NULL ⇒ each Transfer has exactly 1 sender Account
   - an Account can send 0..N Transfers
6. ACCOUNTS 1 —— N TRANSFERS (Receiver role)
   - TRANSFERS.to_account is NOT NULL ⇒ each Transfer has exactly 1 receiver Account
   - an Account can receive 0..N Transfers

4. list of all queries (not the ad hoc ones)

1) Schema Definition (scripts/create_tables.sql)

- **Purpose**: Defines the relational model used by all DAOs/UI flows.
- **Actions**:
  - Creates database BankDB if missing; drops existing tables in dependency-safe order.
  - Creates tables:
    - Customers (identity PK, unique username, national_id, contact info, status).
    - Employees (identity PK, unique username, role, status).
    - Accounts (PK account_number, FK to customer, type, balance, status, dates).
    - Transactions (identity PK, FK to account, type, amount, timestamp, balance_after, reference_code).
    - Transfers (identity PK, from/to account FKs, amount, status, note).
    - Loans (identity PK, FK to account, principal, remaining balance, rate, term, status, due date).
    - OverDraftEvents (identity PK, FK to account, amount, occurred_at, balance_after).
- **Used by**: Initial schema setup; required before seeding or running the app. All DAOs assume these tables exist and match the defined columns.

2) Bulk Seed Data (scripts/seed_data.sql)

- **Purpose**: Populate a rich dataset for demos/testing (100 customers, 300 accounts, 30 loans, transactions, transfers, overdrafts).
- **Actions**:
  - Clears all business tables (OverDraftEvents → Transfers → Transactions → Loans → Accounts → Customers → Employees).
  - Inserts 4 employees with roles (TELLER, LOAN_OFFICER, OPS).
  - Inserts 100 customers using a recursive CTE; generates usernames/pins/national IDs/contact info.

    - Inserts 300 accounts (3 per customer) with rotating account types and deterministic balances/dates.
    - Inserts 30 loans on the first 30 accounts with varying amounts, rates, and statuses.
    - Inserts transactions (2 per first 150 accounts) for seed deposits/withdrawals.
    - Inserts 50 transfers between sequential accounts and mirrors them into transaction logs (TRANSFER_IN/OUT).
    - Inserts 40 overdraft events tied to early accounts.
- **Used by**: Demo environment to exercise UI flows (history, transfers, loans, overdrafts, reports). Provides enough volume for aggregates and filters.

3) Reporting / Analytical Queries (scripts/report_queries.sql)

These queries answer business questions; each includes the SQL and the business perspective.

- **Account inflow/outflow and overdraft risk (aggregate + joins)**
  Business: How much has this account received and paid out, and how risky is it (overdraft count)? Staff use it to assess account health.

```
SELECT
  a.account_number,
  c.name AS customer_name,
  SUM(CASE WHEN t.transaction_type IN ('DEPOSIT','TRANSFER_IN') THEN t.amount ELSE 0 END) AS total_in,
  SUM(CASE WHEN t.transaction_type IN ('WITHDRAWAL','TRANSFER_OUT') THEN t.amount ELSE 0 END) AS total
  COUNT(DISTINCT o.event_id) AS overdraft_events
FROM Accounts a
JOIN Customers c ON c.customer_id = a.customer_id
LEFT JOIN Transactions t ON t.account_number = a.account_number
LEFT JOIN OverDraftEvents o ON o.account_number = a.account_number
WHERE a.account_number = '10000001' -- replace parameter
GROUP BY a.account_number, c.name;
```

- **Overdraft frequency by customer (aggregate + join)**
  Business: Which customers trigger overdrafts, and how often? Highlights customers who may need outreach or account reviews.

```sql
SELECT c.customer_id, c.name, COUNT(o.event_id) AS overdraft_events
FROM Customers c
JOIN Accounts a ON a.customer_id = c.customer_id
LEFT JOIN OverDraftEvents o ON o.account_number = a.account_number
GROUP BY c.customer_id, c.name
HAVING COUNT(o.event_id) > 0;
```

- **High-balance accounts (subquery)**
  Business: Identify accounts above the bank-wide average balance to target premium servicing or retention efforts.

```sql
SELECT account_number, balance
FROM Accounts
WHERE balance > (SELECT AVG(balance) FROM Accounts);
```

- **Loans with above-average remaining balance (subquery)**
  Business: Surface loans with unusually high remaining balances compared to all approved loans, to prioritize monitoring or follow-up.

```sql
SELECT loan_id, account_number, balance_remaining
FROM Loans
WHERE balance_remaining > (
  SELECT AVG(balance_remaining) FROM Loans WHERE status = 'APPROVED'
);
```

- **Customer transaction history (join >2 tables)**
  Business: Produce a statement-like view of all transactions for a customer across accounts, including the customer name for clarity in investigations or support.

```sql
SELECT t.transaction_id, t.timestamp, t.transaction_type, t.amount, a.account_number, c.name
FROM Transactions t
JOIN Accounts a ON a.account_number = t.account_number
JOIN Customers c ON c.customer_id = a.customer_id
WHERE a.customer_id = 1 -- replace parameter
ORDER BY t.timestamp DESC;
```

- **Loan portfolio with ownership (join >2 tables)**
  Business: Show each loan with its status, remaining balance, and who owns it (account + customer), for loan-ops dashboards.

```sql
SELECT l.loan_id, l.status, l.balance_remaining, a.account_number, c.name
FROM Loans l
JOIN Accounts a ON a.account_number = l.account_number
JOIN Customers c ON c.customer_id = a.customer_id
ORDER BY l.start_date DESC;
```

- **Housekeeping: close dormant frozen accounts (conditional UPDATE)**
  Business: Automatically close accounts that are frozen and have zero balance to keep the ledger clean.

```sql
UPDATE Accounts SET status = 'CLOSED' WHERE status = 'FROZEN' AND balance = 0;
```

- **Housekeeping: purge old overdraft events (conditional DELETE with date)**
  Business: Trim overdraft incident history older than the retention window to keep the table lean.

```sql
DELETE FROM OverDraftEvents WHERE occurred_at < DATEADD(day, -30, SYSUTCDATETIME());
```

4) Core Application Queries (DAOs)

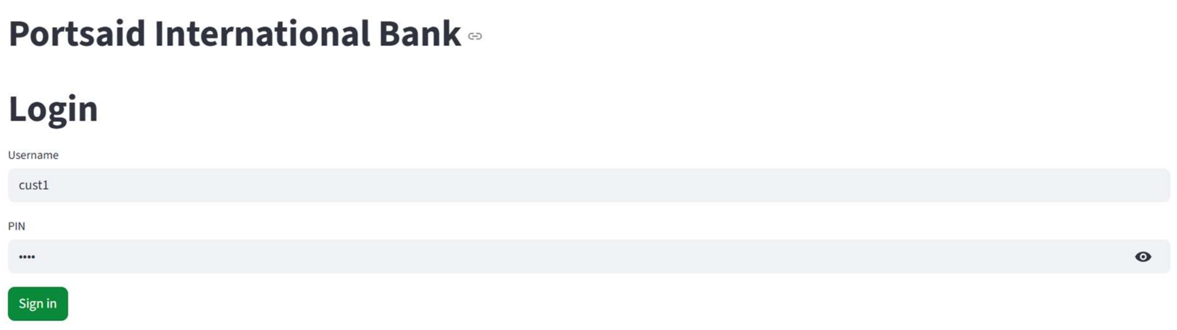
These run via parameterized SQL in the DAOs. Each query is shown with placeholders and its business meaning.

- **Authenticate customer** — Verify active customer credentials to start a customer session.

```sql
SELECT customer_id, name, national_id, email, phone, address, status, pin
FROM Customers
WHERE username = :username AND pin = :pin AND status = 'ACTIVE'
```

-

Screenshot:


Portsaid International Bank

Login

Username

cust1

PIN

••••

- **Create customer** — Register a new customer profile with contact and national ID, set to ACTIVE.

```
INSERT INTO Customers (username, pin, name, email, phone, address, status, national_id)
VALUES (:username, :pin, :name, :email, :phone, :address, 'ACTIVE', :national_id)
```

- **List customers** — Retrieve all customers for employee/admin views.

```
SELECT customer_id, name, national_id, email, phone, address, status, pin
FROM Customers
ORDER BY customer_id ASC
```

EmployeeDAO

- **Authenticate employee** — Verify active employee credentials to start an employee session.

```
SELECT employee_id, username, name, email, phone, role, status
FROM Employees
WHERE username = :username AND pin = :pin AND status = 'ACTIVE'
```

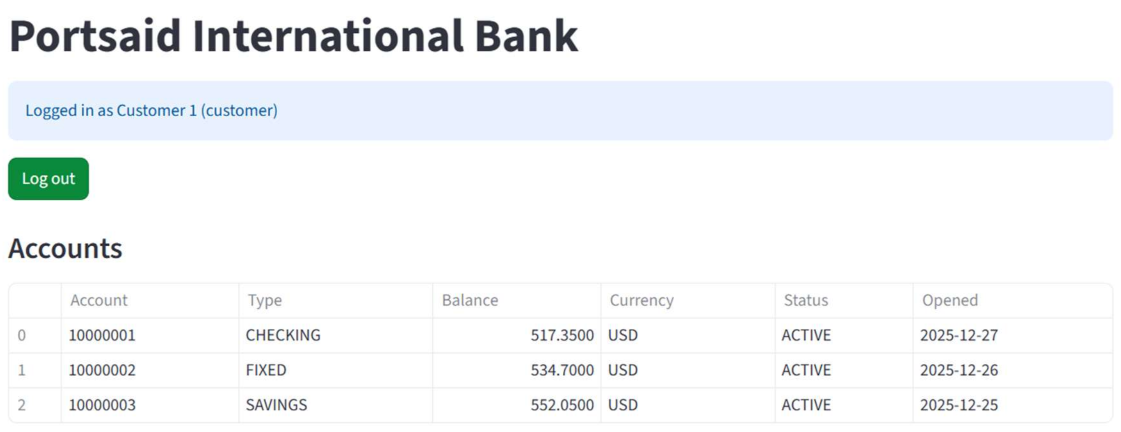
AccountDAO

- **Accounts by customer** — Show all accounts owned by a customer (balances, types, status).

```
SELECT account_number, customer_id, account_type, balance, currency, status, date_opened
FROM Accounts
WHERE customer_id = :customer_id
ORDER BY date_opened DESC
```

-

- Screenshot:



Portsaid International Bank

Logged in as Customer 1 (customer)

Accounts

	Account	Type	Balance	Currency	Status	Opened
0	10000001	CHECKING	517.3500	USD	ACTIVE	2025-12-27
1	10000002	FIXED	534.7000	USD	ACTIVE	2025-12-26
2	10000003	SAVINGS	552.0500	USD	ACTIVE	2025-12-25

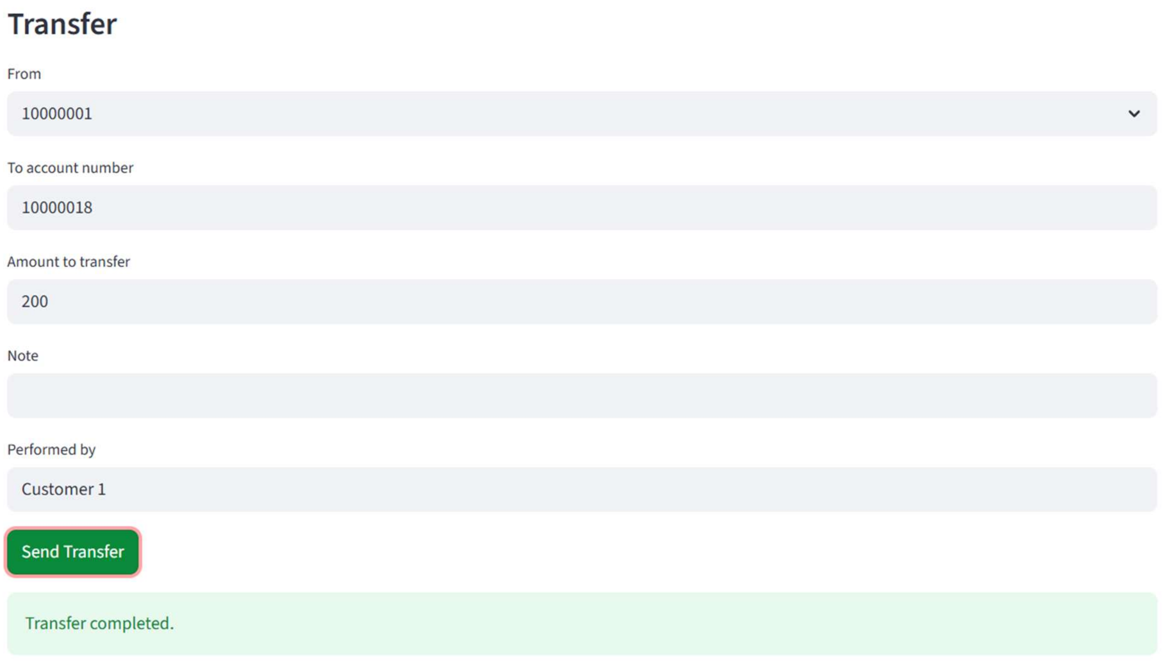
- **Single account** — Fetch one account to validate ownership/status/balance.

```
SELECT account_number, customer_id, account_type, balance, currency, status, date_opened
FROM Accounts
WHERE account_number = :account_number
```

- **Update balance** — Apply balance changes after deposits/withdrawals/transfers.

```
UPDATE Accounts
SET balance = :balance
WHERE account_number = :account_number
```

-

- 
Transfer

From

10000001

To account number

10000018

Amount to transfer

200

Note

Performed by

Customer 1

Transfer completed.

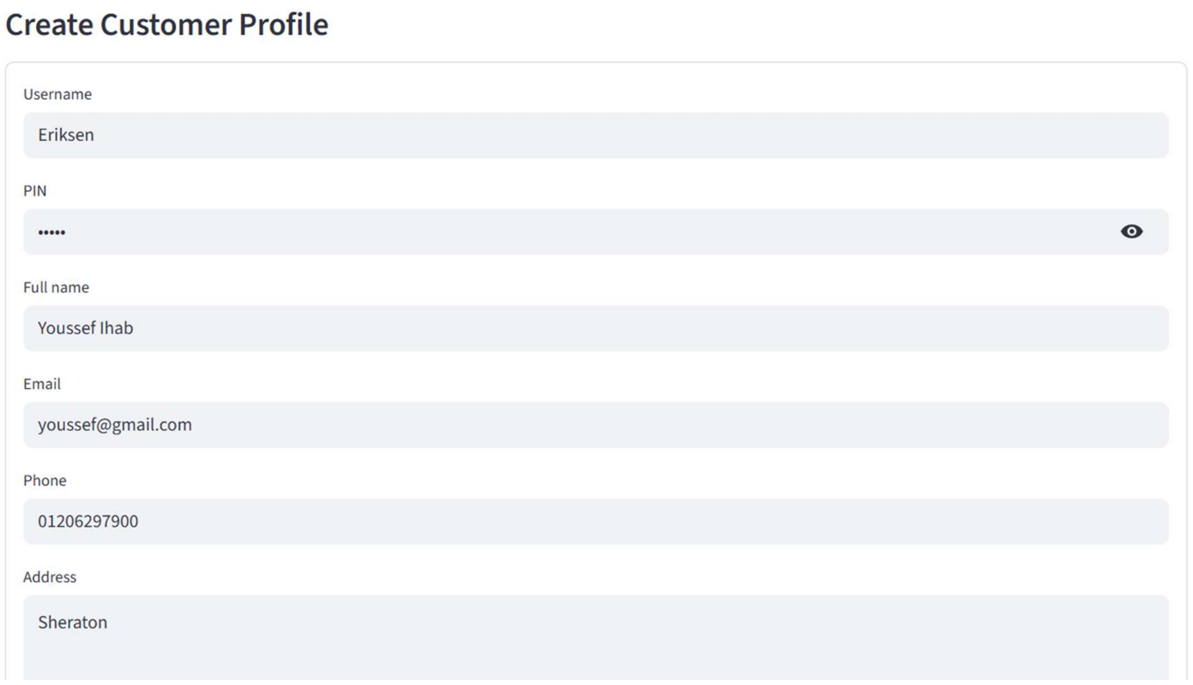
- **Update status** — Freeze/close/activate an account by staff.

```
UPDATE Accounts
SET status = :status
WHERE account_number = :account_number
```

- **Create account** — Open a new account for a customer.

```
INSERT INTO Accounts (account_number, customer_id, account_type, balance, currency, status, date_opened)
VALUES (:account_number, :customer_id, :account_type, :balance, :currency, :status, :date_opened)
```

Screenshot:


Create Customer Profile

Username

Eriksen

PIN

•••••

Full name

Youssef Ihab

Email

youssef@gmail.com

Phone

01206297900

Address

Sheraton

TransactionDAO

- **List history with filters** — Provide statement/history view with optional date/type filters.

SELECT transaction_id, account_number, transaction_type, amount, timestamp, performed_by, note, balance_after, reference_code

FROM Transactions

WHERE <dynamic filters: account_number = :account_number AND optional date/type clauses>

ORDER BY timestamp DESC

- **Insert transaction** — Log any cash/transfer operation with the resulting balance.

INSERT INTO Transactions (account_number, transaction_type, amount, timestamp, performed_by, note, balance_after, reference_code)

OUTPUT INSERTED.transaction_id

VALUES (:account_number, :transaction_type, :amount, SYSUTCDATETIME(), :performed_by, :note, :balance_after, :reference_code)

- **Get transaction by id** — Fetch a specific transaction (post-insert confirmation/audit).

SELECT transaction_id, account_number, transaction_type, amount, timestamp, performed_by, note, balance_after, reference_code

FROM Transactions

WHERE transaction_id = :transaction_id

TransferDAO

- **List transfers for an account** — Show inbound and outbound transfers for an account.

```
SELECT transfer_id, from_account, to_account, amount, timestamp, status, note
FROM Transfers
WHERE from_account = :acct OR to_account = :acct
ORDER BY timestamp DESC
```

- **Insert transfer** — Record a funds move between accounts.

```
INSERT INTO Transfers (from_account, to_account, amount, timestamp, status, note)
OUTPUT INSERTED.transfer_id
VALUES (:from_account, :to_account, :amount, SYSUTCDATETIME(), :status, :note)
```

LoanDAO

- **List loans by account** — Show loans tied to a specific account for the customer view.

SELECT loan_id, account_number, principal, balance_remaining, rate, term_months, start_date, status, next_due_date

FROM Loans

WHERE account_number = :account_number

ORDER BY start_date DESC

- **List all loans** — Portfolio view for employees to review/update.

SELECT loan_id, account_number, principal, balance_remaining, rate, term_months, start_date, status, next_due_date

FROM Loans

ORDER BY start_date DESC

- **Get loan by id** — Retrieve a specific loan for status changes or display.

SELECT loan_id, account_number, principal, balance_remaining, rate, term_months, start_date, status, next_due_date

FROM Loans

WHERE loan_id = :loan_id

- **Request/insert loan** — Create a loan request with initial status (e.g., PENDING).

INSERT INTO Loans (account_number, principal, balance_remaining, rate, term_months, start_date, status)

OUTPUT INSERTED.loan_id

VALUES (:account_number, :principal, :principal, :rate, :term_months, :start_date, :status)

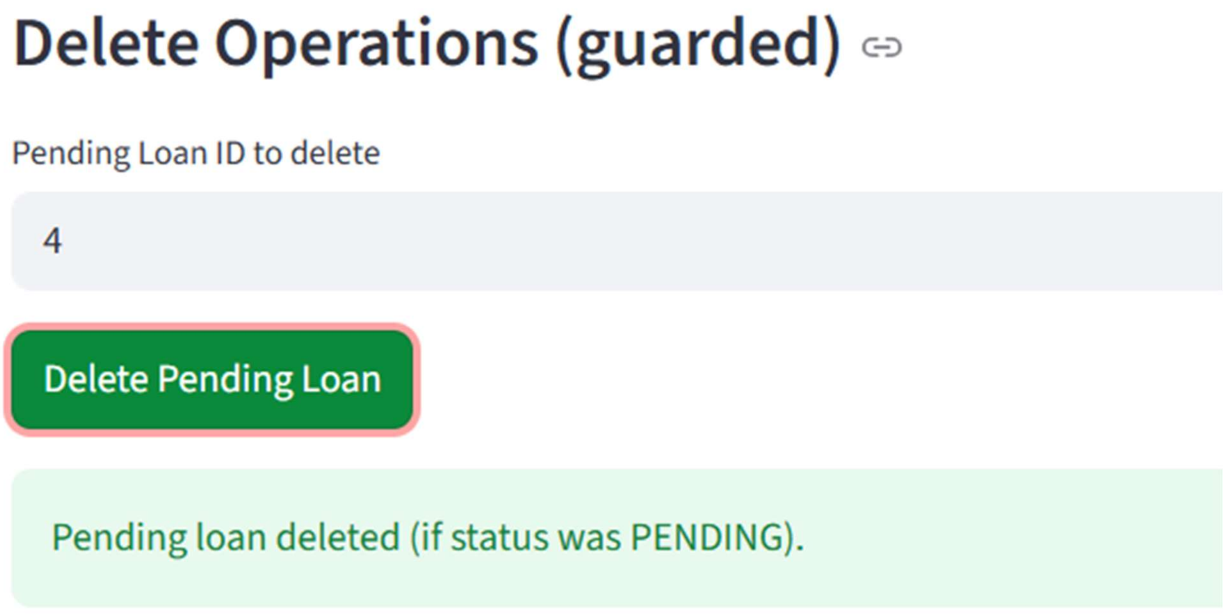
- **Update loan status** — Approve/reject/close a loan.

UPDATE Loans SET status = :status WHERE loan_id = :loan_id

- **Delete pending loan** — Remove only loans that never advanced (PENDING).

DELETE FROM Loans WHERE loan_id = :loan_id AND status = 'PENDING'

Screenshot:



Pending loan deleted (if status was PENDING).

OverDraftEventDAO

- **List overdraft events** — Show overdraft incidents for an account to inform the customer/staff.

SELECT event_id, account_number, amount, occurred_at, note, balance_after

FROM OverDraftEvents

WHERE account_number = :account_number

ORDER BY occurred_at DESC

- **Insert overdraft event** — Log a blocked operation due to insufficient funds.

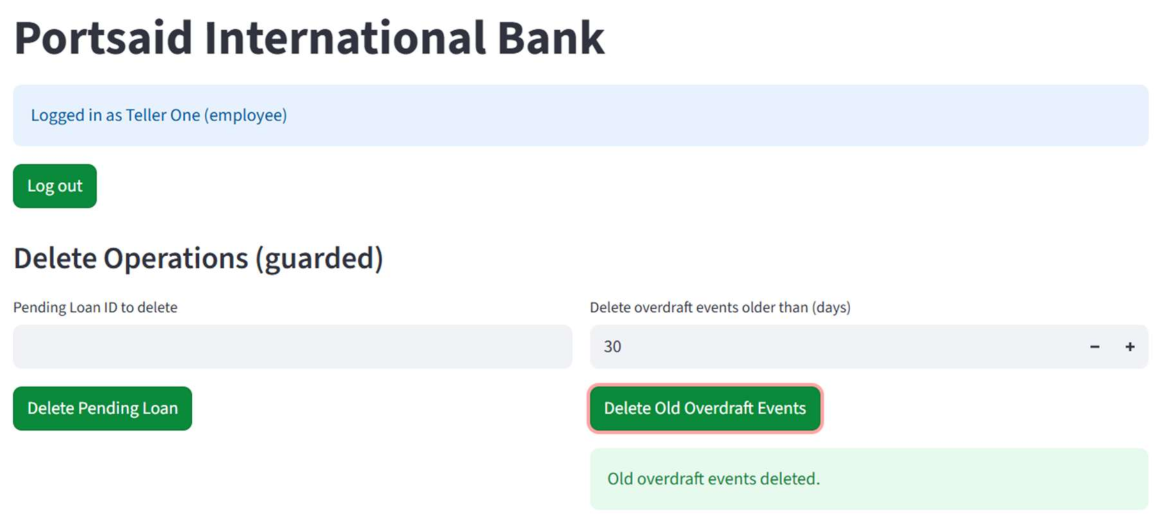
INSERT INTO OverDraftEvents (account_number, amount, occurred_at, note, balance_after)

VALUES (:account_number, :amount, SYSUTCDATETIME(), :note, :balance_after)

- **Delete old overdraft events** — Purge events older than N days per retention.

DELETE FROM OverDraftEvents

WHERE occurred_at < DATEADD(day, -:days, SYSUTCDATETIME())



Portsaid International Bank

Logged in as Teller One (employee)

Log out

Delete Operations (guarded)

Pending Loan ID to delete

Delete overdraft events older than (days)

30

Delete Pending Loan

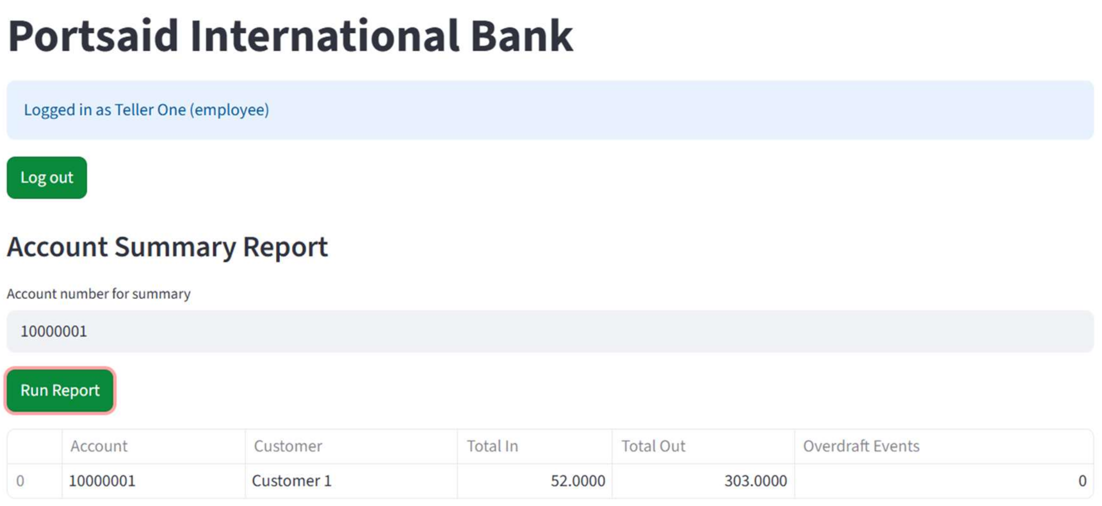
Delete Old Overdraft Events

Old overdraft events deleted.

ReportingDAO

- **Account summary (inflow/outflow/overdrafts)** — Produce a concise health snapshot for an account.

```
SELECT
  a.account_number,
  c.name AS customer_name,
  SUM(CASE WHEN t.transaction_type IN ('DEPOSIT','TRANSFER_IN') THEN t.amount ELSE 0 END) AS total_in,
  SUM(CASE WHEN t.transaction_type IN ('WITHDRAWAL','TRANSFER_OUT') THEN t.amount ELSE 0 END) AS total_out,
  COUNT(DISTINCT o.event_id) AS overdraft_events
FROM Accounts a
JOIN Customers c ON c.customer_id = a.customer_id
LEFT JOIN Transactions t ON t.account_number = a.account_number
LEFT JOIN OverDraftEvents o ON o.account_number = a.account_number
WHERE a.account_number = :account_number
GROUP BY a.account_number, c.name
```



Portsaid International Bank

Logged in as Teller One (employee)

Account Summary Report

Account number for summary

10000001

	Account	Customer	Total In	Total Out	Overdraft Events
0	10000001	Customer 1	52.0000	303.0000	0