# CS 341
# DATABASE SYSTEMS
## RETAIL BANK MANGEMENT SYSTEM

Zara Asim 29262
Hiba Fatima 29005
Aliza M Warris 29224

## Business Scenario

A Retail Bank Management System inspired by Meezan Bank that helps customers perform all basic banking activities in one place

The system will include the following features:

- Customers can deposit money
- Customers can transfer money to other accounts
- Customers can view their transaction statements
- Customers can apply for a locker to store valuables securely
- Customers can apply for a loan from the bank

## Summary of our interview

Our interview with Mr. Mutahir Ahmed provided us with insights into retail banking operations, helping us identify key entities and relationships for our project. The core retail banking functionalities include customer accounts, transactions, and lockers, while loans and personal loans are excluded. Services such as ATMs, cards, and remittances are linked to accounts but function separately. Customers interact with accounts for deposits, withdrawals, transfers, and locker services.

## Business Rules

## Customer and User Authentication

| Area | Rules |
|---|---|
| **Customer Identity** | CNIC must be unique and exactly 13 digits. |
| **Age Eligibility** | Customer must be at least 18 years old |
| **Contact** | PHONE_NUMBER must be 11 digits if provided. |
| **Email** | EMAIL must be unique. |
| **Account Ownership** | A customer may be linked to one or many accounts through ACCOUNT_HOLDER. |
| **Authentication** | Each CUSTOMER can have **only one USER_AUTH** record (CUSTOMER_ID UNIQUE in USER_AUTH). |
| **Password Security** | USER_AUTH must store password as PASSWORD_HASH — plain passwords not allowed. |

| | |
|---|---|
| **Auth Status** | USER_AUTH.STATUS can be Active, Locked, or Suspended. |

## Account Management

| Area | Rules |
|---|---|
| **Account** | ACCOUNT_NUMBER must be unique. |
| **Branch Relation** | Every ACCOUNT must belong to an existing BRANCH. |
| **Account Type** | Every ACCOUNT must reference a valid ACCOUNT_TYPE. |
| **Holders** | ACCOUNT_HOLDER must specify HOLDER_TYPE which can be either **Primary, Secondary**, or **Joint** |
| **Mode & Status** | ACCOUNT_MODE must be **Individua**l or **Joint** and ACCOUNT.STATUS must be **Active, Dormant, Frozen,** or **Closed**. |
| **Dates** | OPENED_DATE cannot be in the future. |
| **Balance** | ACCOUNT.BALANCE must always be **≥ 0** and updated after every transaction. |
| **Closed Accounts** | Closed accounts must not accept new transactions |

## Transaction Management

| Area | Rules |
|---|---|
| **Valid Account** | Every transaction must be linked to an existing ACCOUNT via ACCOUNT_ID |
| **Date Constraint** | TRANSACTION_DATE cannot be set in the future. |
| **Types** | TRANSACTION_TYPE must be either **Deposit, Withdrawal, Transfer, Fee**. |
| **Modes** | TRANSACTION_MODE must be one of: **Cash, Cheque, Online, Mobile** |
| **Amount** | AMOUNT must be greater than 0. |
| **Balance Tracking** | BALANCE_REMAINING must match previous balance ± AMOUNT depending on transaction type. |
| **Insufficient Funds** | 'Withdrawal' or 'Transfer' cannot occur if balance is insufficient. |

## Card Management

| Area | Rules |
|---|---|
| **Relation** | CARD must be linked to an existing ACCOUNT. |
| **Format** | CARD_NUMBER must be unique and exactly 16 digits. |
| **Expiry** | EXPIRY_DATE must be later than ISSUED_DATE. |
| **Status** | CARD.STATUS must be one of: **Active, Blocked, Expired, Lost, Stolen**. |
| **Limit** | DAILY_LIMIT must be a positive number. |

## Loan Management

| Area | Rules |
|---|---|
| **Loan Type** | • TYPE_NAME must be unique and must be one of : **Housing, Car**<br>• PROFIT_RATE > 0 |
| **Loan Application** | • Must reference CUSTOMER, BRANCH, and LOAN_TYPE.<br>• REQUESTED_AMOUNT must be > 0 |
| **Loan Application Status** | STATUS should be one of: **Pending, Under Review, Approved, Rejected** |

## Locker Management

| Area | Rules |
|---|---|
| **Locker Assignment** | LOCKER must belong to a BRANCH. |
| **Uniqueness** | Combination (locker_number + branch_id) must be unique. |

| | |
|---|---|
| **Locker Status** | STATUS could be one of: **Available, Occupied, Under Maintenance** |
| **Locker Rental** | Must reference both LOCKER and ACCOUNT. |
| **Date Constraint** | RENTAL end_date must be greater than start_date. |
| **Occupancy Transition** | • On new rental, Locker becomes **Occupied**<br>• On rental expiration/closing, Locker returns to **Available** |

## Entities, Attributes, and Relationships

Entity: CUSTOMER

Attributes: CUSTOMER_ID (PK), FULL_NAME, CNIC, DATE_OF_BIRTH, PHONE_NUMBER, EMAIL, ADDRESS

Relationships with Multiplicity Constraint

| Relationships | Multiplicity |
|---|---|
| CUSTOMER hold ACCOUNT (via ACCOUNT_HOLDER) | M:M |
| CUSTOMER has USER_AUTH | 1:1 |
| CUSTOMER applies LOAN_APPLICATION | 1:M |
| CUSTOMER rents LOCKER_RENTAL (VIA ACCOUNT) | M:M |

Entity: USER_AUTH

Attributes: USER_AUTH_ID (PK), CUSTOMER_ID (FK), USERNAME, PASSWORD_HASH, STATUS

Relationships with Multiplicity Constraint

| Relationships | Multiplicity |
|---|---|
| USER_AUTH belongs to CUSTOMER | 1:1 |

Entity: BRANCH

Attributes: BRANCH_ID (PK), NAME, LOCATION

Relationships with Multiplicity Constraint

| Relationships | Multiplicity |
|---|---|
| BRANCH manages ACCOUNT | 1:M |
| BRANCH manages LOAN_APPLICATION | 1:M |
| BRANCH manages LOAN_ACCOUNT | 1:M |
| BRANCH owns LOCKER | 1:M |

Entity: ACCOUNT_TYPE

Attributes: ACCOUNT_TYPE_ID (PK), TYPE_NAME, MIN_BALANCE, MONTHLY_FEE

Relationships with Multiplicity Constraint

| Relationships | Multiplicity |
|---|---|
| ACCOUNT_TYPE categorizes ACCOUNT | 1:M |

Entity: ACCOUNT

Attributes: ACCOUNT_ID (PK), ACCOUNT_NUMBER, TYPE_ID (FK), BRANCH_ID (FK), BALANCE, ACCOUNT_MODE, STATUS, OPENED_DATE

Relationships with Multiplicity Constraint

| Relationships | Multiplicity |
|---|---|
| ACCOUNT is held by CUSTOMER (via ACCOUNT_HOLDER) | M:M |
| ACCOUNT is linked to ACCOUNT_TYPE | M:1 |
| ACCOUNT is located at BRANCH | M:1 |
| ACCOUNT performs BANK_TRANSACTION | 1:M |
| ACCOUNT rents LOCKER_RENTAL | 1:M |
| ACCOUNT owns CARD | 1:M |

Entity: ACCOUNT_HOLDER

Attributes: ACCOUNT_ID (FK), CUSTOMER_ID (FK), HOLDER_TYPE

Relationships with Multiplicity Constraint

| Relationships | Multiplicity |
|---|---|

| | |
|---|---|
| ACCOUNT_HOLDER has an account | 1: M |

Entity: BANK_TRANSACTION

Attributes: TRANSACTION_ID (PK), ACCOUNT_ID (FK), AMOUNT, TRANSACTION_TYPE, TRANSACTION_MODE, TRANSACTION_DATE, BALANCE_REMAINING

| Relationships | Multiplicity |
|---|---|
| BANK_TRANSACTION is linked to ACCOUNT | M:1 |

Entity: CARD

Attributes: CARD_ID (PK), ACCOUNT_ID (FK), CARD_NUMBER, CARD_TYPE, EXPIRY_DATE, ISSUED_DATE, STATUS, DAILY_LIMIT

Relationships with Multiplicity Constraint

| Relationships | Multiplicity |
|---|---|
| CARD issued for ACCOUNT | M:1 |

Entity: LOAN_TYPE

Attributes: LOAN_TYPE_ID (PK), TYPE_NAME, PROFIT_RATE, MIN_AMOUNT, MAX_AMOUNT, MAX_DURATION_MONTHS

Relationships with Multiplicity Constraint

| Relationships | Multiplicity |
|---|---|
| LOAN_TYPE categorizes LOAN_APPLICATION | 1:M |
| LOAN_TYPE categorizes LOAN_ACCOUNT | 1:M |

Entity: LOAN_APPLICATION

Attributes: APPLICATION_ID (PK), CUSTOMER_ID (FK), BRANCH_ID (FK), LOAN_TYPE_ID (FK), REQUESTED_AMOUNT, APPLICATION_DATE, STATUS

Relationships with Multiplicity Constraint

| Relationships | Multiplicity |
|---|---|
| LOAN_APPLICATION submitted by CUSTOMER | M:1 |
| LOAN_APPLICATION processes by BRANCH | M:1 |
| LOAN_APPLICATION refers to LOAN_TYPE | M:1 |

Entity: LOCKER

Attributes: LOCKER_ID (PK), BRANCH_ID (FK), LOCKER_NUMBER, LOCKER_SIZE, ANNUAL_FEE, STATUS

Relationships with Multiplicity Constraint

| Relationships | Multiplicity |
|---|---|
| LOCKER located at BRANCH | M:1 |
| LOCKER rented via LOCKER_RENTAL | 1:M |

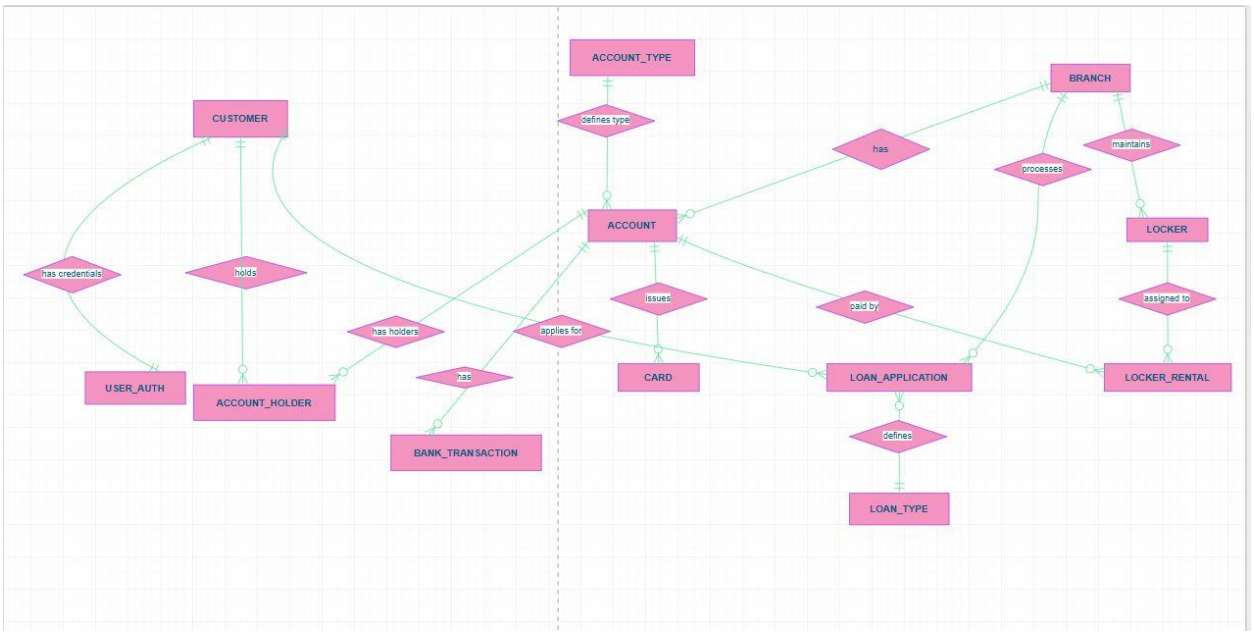
Entity: LOCKER_RENTAL

Attributes: RENTAL_ID (PK), LOCKER_ID (FK), ACCOUNT_ID (FK), START_DATE, END_DATE, STATUS

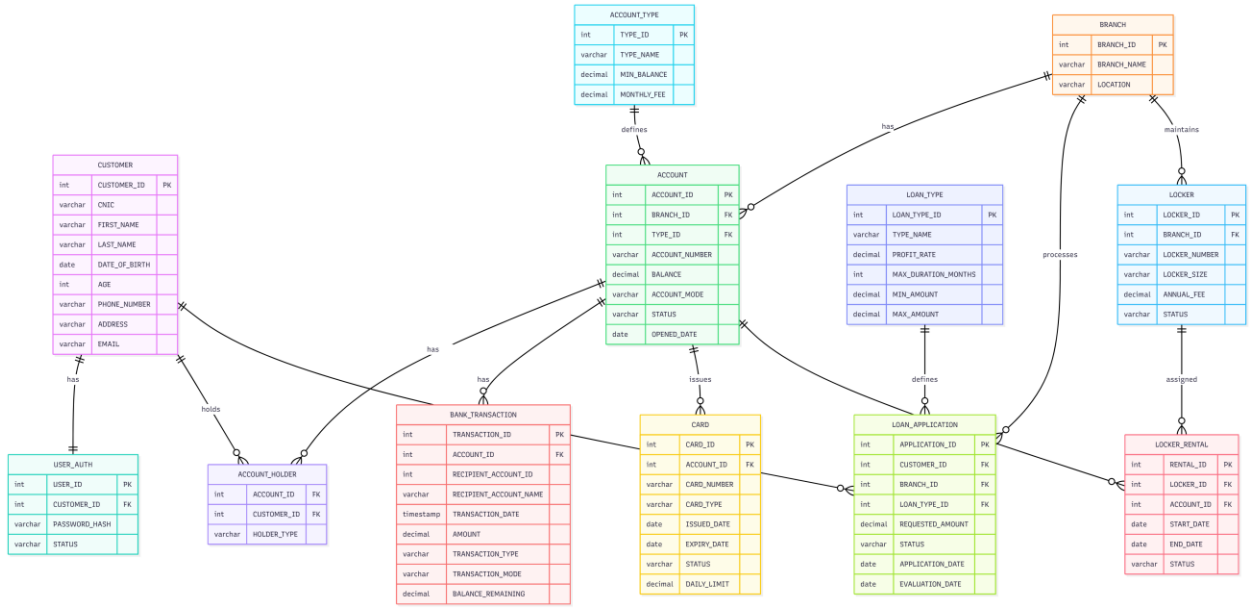
Relationships with Multiplicity Constraint

| Relationships | Multiplicity |
|---|---|
| LOCKER_RENTAL rented by ACCOUNT | M:1 |
| LOCKER_RENTAL assigned to LOCKER | M:1 |

## ER diagram




## Relational Schema




## Normalization up to 3rd Normal Form (3NF)

## 0NF (Unnormalized Form)

BANK_DETAILS( Untitled-1 ●

```
BANK_DETAILS(
  CUSTOMER_ID, CNIC, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, AGE,
  PHONE_NUMBER, ADDRESS, EMAIL,
  USER_ID, PASSWORD_HASH, USER_STATUS,
  BRANCH_ID, BRANCH_NAME, LOCATION,
  TYPE_ID, TYPE_NAME, MIN_BALANCE, MONTHLY_FEE,
  ACCOUNT_ID, ACCOUNT_NUMBER, BALANCE, ACCOUNT_MODE,
  ACCOUNT_STATUS, OPENED_DATE,
  HOLDER_TYPE,
  TRANSACTION_ID, RECIPIENT_ACCOUNT_ID, RECIPIENT_ACCOUNT_NAME,
  TRANSACTION_DATE, AMOUNT, TRANSACTION_TYPE,
  TRANSACTION_MODE, BALANCE_REMAINING,
  CARD_ID, CARD_NUMBER, CARD_TYPE, CARD_ISSUED_DATE,
  CARD_EXPIRY_DATE, CARD_STATUS, DAILY_LIMIT,
  LOAN_TYPE_ID, LOAN_TYPE_NAME, PROFIT_RATE,
  MAX_DURATION_MONTHS, MIN_AMOUNT, MAX_AMOUNT,
  APPLICATION_ID, REQUESTED_AMOUNT, APPLICATION_STATUS,
  APPLICATION_DATE, EVALUATION_DATE,
  LOCKER_ID, LOCKER_NUMBER, LOCKER_SIZE,
  ANNUAL_FEE, LOCKER_STATUS,
  RENTAL_ID, RENTAL_START_DATE, RENTAL_END_DATE, RENTAL_STATUS
)
```

## 1NF (First Normal Form)

Rules applied:

- Identify primary key: **(CUSTOMER_ID, ACCOUNT_ID)**
- All attributes are atomic
- Identify partial and transitive dependencies

BANK_DETAILS_1NF( Untitled-1 ●

```
BANK_DETAILS_1NF(
  CUSTOMER_ID, ACCOUNT_ID,
  CNIC, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, AGE,
  PHONE_NUMBER, ADDRESS, EMAIL,
  USER_ID, PASSWORD_HASH, USER_STATUS,
  BRANCH_ID, BRANCH_NAME, LOCATION,
  TYPE_ID, TYPE_NAME, MIN_BALANCE, MONTHLY_FEE,
  ACCOUNT_NUMBER, BALANCE, ACCOUNT_MODE,
  ACCOUNT_STATUS, OPENED_DATE,
  HOLDER_TYPE,
  TRANSACTION_ID, RECIPIENT_ACCOUNT_ID, RECIPIENT_ACCOUNT_NAME,
  TRANSACTION_DATE, AMOUNT, TRANSACTION_TYPE,
  TRANSACTION_MODE, BALANCE_REMAINING,
  CARD_ID, CARD_NUMBER, CARD_TYPE, CARD_ISSUED_DATE,
  CARD_EXPIRY_DATE, CARD_STATUS, DAILY_LIMIT,
  LOAN_TYPE_ID, LOAN_TYPE_NAME, PROFIT_RATE,
  MAX_DURATION_MONTHS, MIN_AMOUNT, MAX_AMOUNT,
  APPLICATION_ID, REQUESTED_AMOUNT, APPLICATION_STATUS,
  APPLICATION_DATE, EVALUATION_DATE,
  LOCKER_ID, LOCKER_NUMBER, LOCKER_SIZE,
  ANNUAL_FEE, LOCKER_STATUS,
  RENTAL_ID, RENTAL_START_DATE, RENTAL_END_DATE, RENTAL_STATUS
)
Primary Key: (CUSTOMER_ID, ACCOUNT_ID)
```

Untitled-1 ● Partial Dependency Untitled-2 ●

```
Partial Dependency
CUSTOMER_ID → (CNIC, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, AGE,
               PHONE_NUMBER, ADDRESS, EMAIL, USER_ID, PASSWORD_HASH,
               USER_STATUS, APPLICATION_ID, REQUESTED_AMOUNT,
               APPLICATION_STATUS, APPLICATION_DATE, EVALUATION_DATE,
               LOAN_TYPE_ID, LOAN_TYPE_NAME, PROFIT_RATE,
               MAX_DURATION_MONTHS, MIN_AMOUNT, MAX_AMOUN)

ACCOUNT_ID → (BRANCH_ID, BRANCH_NAME, LOCATION, TYPE_ID, TYPE_NAME,
              MIN_BALANCE, MONTHLY_FEE, ACCOUNT_NUMBER, BALANCE,
              ACCOUNT_MODE, ACCOUNT_STATUS, OPENED_DATE, TRANSACTION_ID,
              RECIPIENT_ACCOUNT_ID, RECIPIENT_ACCOUNT_NAME, TRANSACTION_DATE,
              AMOUNT, TRANSACTION_TYPE, TRANSACTION_MODE, BALANCE_REMAINING,
              CARD_ID, CARD_NUMBER, CARD_TYPE, CARD_ISSUED_DATE,
              CARD_EXPIRY_DATE, CARD_STATUS, DAILY_LIMIT, LOCKER_ID,
              LOCKER_NUMBER, LOCKER_SIZE, ANNUAL_FEE, LOCKER_STATUS,
              RENTAL_ID, RENTAL_START_DATE, RENTAL_END_DATE, RENTAL_STATUS)

(CUSTOMER_ID, ACCOUNT_ID) → (HOLDER_TYPE)
```

Transitive Dependency Untitled-1 ● | Partial Dependency Untitled-2 ●

```
Transitive Dependency
USER_ID → (PASSWORD_HASH, USER_STATUS)

LOAN_TYPE_ID → (LOAN_TYPE_NAME, PROFIT_RATE, MAX_DURATION_MONTHS,
                MIN_AMOUNT, MAX_AMOUNT)

APPLICATION_ID → (BRANCH_ID, LOAN_TYPE_ID, REQUESTED_AMOUNT,
                  APPLICATION_STATUS, APPLICATION_DATE, EVALUATION_DATE)

BRANCH_ID → (BRANCH_NAME, LOCATION)

BRANCH_ID → (BRANCH_NAME, LOCATION)

TYPE_ID → (TYPE_NAME, MIN_BALANCE, MONTHLY_FEE)

TRANSACTION_ID → (RECIPIENT_ACCOUNT_ID, RECIPIENT_ACCOUNT_NAME,
                  TRANSACTION_DATE, AMOUNT, TRANSACTION_TYPE,
                  TRANSACTION_MODE, BALANCE_REMAINING)

CARD_ID → (CARD_NUMBER, CARD_TYPE, CARD_ISSUED_DATE,
           CARD_EXPIRY_DATE, CARD_STATUS, DAILY_LIMIT)

LOCKER_ID → (LOCKER_NUMBER, LOCKER_SIZE, ANNUAL_FEE,
             LOCKER_STATUS)

RENTAL_ID → (LOCKER_ID, RENTAL_START_DATE, RENTAL_END_DATE,
             RENTAL_STATUS)
```

## 2NF (Second Normal Form)

Rules applied:

- Should be in 1NF
- Remove partial dependencies

2NF Untitled-3 ● | Transitive Dependency Untitled-1 ● | Partial Dependency Untitled-2

```
2NF
CUSTOMER(
  CUSTOMER_ID,
  CNIC, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, AGE,
  PHONE_NUMBER, ADDRESS, EMAIL, USER_ID, PASSWORD_HASH,
  USER_STATUS, APPLICATION_ID, REQUESTED_AMOUNT,
  APPLICATION_STATUS, APPLICATION_DATE, EVALUATION_DATE,
  BRANCH_ID, LOAN_TYPE_ID, LOAN_TYPE_NAME, PROFIT_RATE,
  MAX_DURATION_MONTHS, MIN_AMOUNT, MAX_AMOUNT
)
Primary Key: CUSTOMER_ID

ACCOUNT(
  ACCOUNT_ID,
  BRANCH_ID, BRANCH_NAME, LOCATION, TYPE_ID, TYPE_NAME,
  MIN_BALANCE, MONTHLY_FEE, ACCOUNT_NUMBER, BALANCE,
  ACCOUNT_MODE, ACCOUNT_STATUS, OPENED_DATE, TRANSACTION_ID,
  RECIPIENT_ACCOUNT_ID, RECIPIENT_ACCOUNT_NAME, TRANSACTION_DATE,
  AMOUNT, TRANSACTION_TYPE, TRANSACTION_MODE, BALANCE_REMAINING,
  CARD_ID, CARD_NUMBER, CARD_TYPE, CARD_ISSUED_DATE,
  CARD_EXPIRY_DATE, CARD_STATUS, DAILY_LIMIT, RENTAL_ID,
  LOCKER_ID, LOCKER_NUMBER, LOCKER_SIZE, ANNUAL_FEE,
  LOCKER_STATUS, RENTAL_START_DATE, RENTAL_END_DATE,
  RENTAL_STATUS
)
Primary Key: ACCOUNT_ID

ACCOUNT_HOLDER(
  CUSTOMER_ID, ACCOUNT_ID,
  HOLDER_TYPE
)
Primary Key: (CUSTOMER_ID, ACCOUNT_ID)
Foreign Keys: CUSTOMER_ID → CUSTOMER, ACCOUNT_ID → ACCOUNT
```

## 3NF (Third Normal Form)

Rules applied:

- Should be in 2NF
- Remove transitive dependencies

3NF Untitled-4 ● | 2NF Untitled-3 ● | Transitive Dependency Untitled-1 ●

```
3NF

CUSTOMER(
  CUSTOMER_ID,
  CNIC, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, AGE,
  PHONE_NUMBER, ADDRESS, EMAIL
)
Primary Key: CUSTOMER_ID

USER_AUTH(
  USER_ID,
  CUSTOMER_ID, PASSWORD_HASH, USER_STATUS
)
Primary Key: USER_ID
Foreign Key: CUSTOMER_ID → CUSTOMER

BRANCH(
  BRANCH_ID,
  BRANCH_NAME, LOCATION
)
Primary Key: BRANCH_ID

ACCOUNT_TYPE(
  TYPE_ID,
  TYPE_NAME, MIN_BALANCE, MONTHLY_FEE
)
Primary Key: TYPE_ID
```

3NF Untitled-4 ● | 2NF Untitled-3 ● | Transitive Dependency Untitled-1 ● | Part

```
LOAN_TYPE(
  LOAN_TYPE_ID,
  LOAN_TYPE_NAME, PROFIT_RATE, MAX_DURATION_MONTHS,
  MIN_AMOUNT, MAX_AMOUNT
)
Primary Key: LOAN_TYPE_ID

ACCOUNT(
  ACCOUNT_ID,
  BRANCH_ID, TYPE_ID, ACCOUNT_NUMBER, BALANCE,
  ACCOUNT_MODE, ACCOUNT_STATUS, OPENED_DATE
)
Primary Key: ACCOUNT_ID
Foreign Keys: BRANCH_ID → BRANCH, TYPE_ID → ACCOUNT_TYPE

ACCOUNT_HOLDER(
  CUSTOMER_ID, ACCOUNT_ID,
  HOLDER_TYPE
)
Primary Key: (CUSTOMER_ID, ACCOUNT_ID)
Foreign Keys: CUSTOMER_ID → CUSTOMER, ACCOUNT_ID → ACCOUNT

LOAN_APPLICATION(
  APPLICATION_ID,
  CUSTOMER_ID, BRANCH_ID, LOAN_TYPE_ID, REQUESTED_AMOUNT,
  APPLICATION_STATUS, APPLICATION_DATE, EVALUATION_DATE
)
Primary Key: APPLICATION_ID
Foreign Keys: CUSTOMER_ID → CUSTOMER, BRANCH_ID → BRANCH,
              LOAN_TYPE_ID → LOAN_TYPE
```

3NF Untitled-4 ● | 2NF Untitled-3 ● | Transitive Dependency Untitled-1 ●

```
BANK_TRANSACTION(
  TRANSACTION_ID,
  ACCOUNT_ID, RECIPIENT_ACCOUNT_ID, RECIPIENT_ACCOUNT_NAME,
  TRANSACTION_DATE, AMOUNT, TRANSACTION_TYPE,
  TRANSACTION_MODE, BALANCE_REMAINING
)
Primary Key: TRANSACTION_ID
Foreign Key: ACCOUNT_ID → ACCOUNT

CARD(
  CARD_ID,
  ACCOUNT_ID, CARD_NUMBER, CARD_TYPE, CARD_ISSUED_DATE,
  CARD_EXPIRY_DATE, CARD_STATUS, DAILY_LIMIT
)
Primary Key: CARD_ID
Foreign Key: ACCOUNT_ID → ACCOUNT

LOCKER(
  LOCKER_ID,
  BRANCH_ID, LOCKER_NUMBER, LOCKER_SIZE,
  ANNUAL_FEE, LOCKER_STATUS
)
Primary Key: LOCKER_ID
Foreign Key: BRANCH_ID → BRANCH

LOCKER_RENTAL(
  RENTAL_ID,
  LOCKER_ID, ACCOUNT_ID, RENTAL_START_DATE,
  RENTAL_END_DATE, RENTAL_STATUS
)
Primary Key: RENTAL_ID
Foreign Keys: LOCKER_ID → LOCKER, ACCOUNT_ID → ACCOUNT
```

## DDL script screenshots

## Constraints Applied to Our Schema

## 1. CUSTOMER TABLE

| Constraint Type | Column | Explanation |
|---|---|---|
| **PRIMARY KEY** | CUSTOMER_ID | Ensures that each customer has a unique ID. |
| **UNIQUE** | CNIC | No two customers can share the same CNIC. |
| **CHECK** | LENGTH(CNIC) = 13 | CNIC must always be 13 digits. |
| **NOT NULL** | FIRST_NAME, LAST_NAME, DATE_OF_BIRTH | Customer names and DOB are mandatory. |
| **CHECK** | LENGTH(PHONE_NUMBER) = 11 | Phone number must be exactly 11 digits (Pakistani format). |
| **UNIQUE** | EMAIL | No two customers can have the same email. |
| **AGE** | Calculated by trigger | AGE is automatically computed; the user cannot manually change it. |

## 2. USER_AUTH TABLE

| Constraint Type | Column | Explanation |
|---|---|---|
| **PRIMARY KEY** | USER_ID | Unique user login identity. |
| **UNIQUE** | CUSTOMER_ID | One customer can have *only one* user account. |
| **NOT NULL** | PASSWORD_HASH | Passwords are mandatory. |
| **CHECK** | STATUS IN ('Active','Locked','Suspended') | Prevents invalid status values. |
| **FOREIGN KEY** | CUSTOMER_ID → CUSTOMER(CUSTOMER_ID) | Links login with an existing customer. |
| **ON DELETE CASCADE** | | If a customer is deleted, their login is also deleted automatically. |

## 3. BRANCH TABLE

| Constraint Type | Column | Explanation |
|---|---|---|
| **PRIMARY KEY** | BRANCH_ID | Unique branch identity. |

| | | |
|---|---|---|
| **UNIQUE** | BRANCH_NAME | No duplicate branch names. |
| **NOT NULL** | BRANCH_NAME | A branch must have a name. |

## 4. ACCOUNT_TYPE TABLE

| Constraint Type | Column | Explanation |
|---|---|---|
| **PRIMARY KEY** | TYPE_ID | Unique account type (Savings/Current/Islamic). |
| **UNIQUE** | TYPE_NAME | No duplicate names for account types. |
| **DEFAULT** | MIN_BALANCE, MONTHLY_FEE | Automatically sets base values. |

## 5. ACCOUNT TABLE

| Constraint Type | Column | Explanation |
|---|---|---|
| **PRIMARY KEY** | ACCOUNT_ID | Unique bank account. |
| **UNIQUE** | ACCOUNT_NUMBER | Prevents duplicate account numbers. |
| **CHECK** | BALANCE ≥ 0 | Account balance cannot be negative. |
| **CHECK** | ACCOUNT_MODE IN ('Individual','Joint') | Only valid account modes allowed. |
| **CHECK** | STATUS IN ('Active','Dormant','Frozen','Closed') | Ensures correct account life cycle. |
| **DEFAULT** | OPENED_DATE = SYSDATE | Store automatically when an account is created. |
| **FOREIGN KEY** | BRANCH_ID → BRANCH(BRANCH_ID) | Every account must belong to a branch. |
| **FOREIGN KEY** | TYPE_ID → ACCOUNT_TYPE(TYPE_ID) | Every account must have a type. |

## 6. ACCOUNT_HOLDER TABLE

| Constraint Type | Column | Explanation |
|---|---|---|
| **PRIMARY KEY** | (ACCOUNT_ID, CUSTOMER_ID) | Prevents duplicate mapping. |
| **CHECK** | HOLDER_TYPE IN ('Primary','Secondary','Joint') | Only valid holder roles allowed. |
| **FOREIGN KEY** | ACCOUNT_ID → ACCOUNT | Deletes holder record if account is deleted (CASCADE). |
| **FOREIGN KEY** | CUSTOMER_ID → CUSTOMER | Deletes holder record if customer is deleted (CASCADE). |

## 7. BANK_TRANSACTION TABLE

| Constraint Type | Column | Explanation |
|---|---|---|
| **PRIMARY KEY** | TRANSACTION_ID | Unique for each transaction. |
| **NOT NULL** | AMOUNT | Amount is mandatory. |
| **CHECK** | AMOUNT > 0 | You cannot send 0 or negative money. |
| **CHECK** | TRANSACTION_TYPE IN ('Deposit','Transfer','Fee','Withdrawal') | Ensures only valid transaction types are used. |
| **CHECK** | TRANSACTION_MODE IN ('Cash','Cheque','Online','Mobile') | Valid transaction channels only. |
| **FOREIGN KEY** | ACCOUNT_ID → ACCOUNT | Every transaction must belong to a valid account. |
| **ON DELETE CASCADE** | | All transactions are deleted if account is deleted. |

## 8. CARD TABLE

| Constraint Type | Column | Explanation |
|---|---|---|
| **PRIMARY KEY** | CARD_ID | Unique card ID. |
| **UNIQUE** | CARD_NUMBER | Prevents duplicate debit/credit card numbers. |
| **CHECK** | LENGTH(CARD_NUMBER) = 16 | Ensures card number length is correct. |
| **CHECK** | CARD_TYPE IN ('Debit','Credit','Prepaid') | Valid card categories only. |
| **CHECK** | STATUS IN ('Active','Blocked','Expired','Lost','Stolen') | Ensures card status accuracy. |
| **CHECK** | DAILY_LIMIT > 0 | Card spending limit must be positive. |
| **CHECK** | EXPIRY_DATE > ISSUED_DATE | Prevents expired cards at creation. |
| **FOREIGN KEY** | ACCOUNT_ID → ACCOUNT | Every card must belong to an account. |

## 9. LOAN_TYPE TABLE

| Constraint Type | Column | Explanation |
|---|---|---|
| **PRIMARY KEY** | LOAN_TYPE_ID | Unique loan category. |
| **UNIQUE** | TYPE_NAME | No duplicate loan types. |
| **CHECK** | PROFIT_RATE > 0 | Profit must be positive. |
| **CHECK** | MAX_DURATION_MONTHS > 0 | Loan must have a valid duration. |

## 10. LOAN_APPLICATION TABLE

| Constraint Type | Column | Explanation |
|---|---|---|
| **PRIMARY KEY** | APPLICATION_ID | Unique loan application. |
| **CHECK** | REQUESTED_AMOUNT > 0 | Loan amount must be positive. |
| **CHECK** | STATUS IN ('Pending','Under Review','Approved','Rejected') | Valid application states only. |
| **DEFAULT** | APPLICATION_DATE = SYSDATE | Automatically stores request date. |
| **FOREIGN KEY** | CUSTOMER_ID → CUSTOMER | Loan belongs to an existing customer. |
| **FOREIGN KEY** | BRANCH_ID → BRANCH | Loan is taken from an existing branch. |
| **FOREIGN KEY** | LOAN_TYPE_ID → LOAN_TYPE | Loan must be of a defined type. |

## 11. LOCKER TABLE

| Constraint Type | Column | Explanation |
|---|---|---|
| **PRIMARY KEY** | LOCKER_ID | Unique locker ID. |
| **CHECK** | LOCKER_SIZE IN ('Small','Medium','Large') | Prevents wrong sizes. |
| **CHECK** | STATUS IN ('Available','Occupied','Under Maintenance') | Valid locker life-cycle states. |
| **UNIQUE** | (BRANCH_ID, LOCKER_NUMBER) | Same locker number cannot repeat in same branch. |
| **FOREIGN KEY** | BRANCH_ID → BRANCH | Locker belongs to a branch. |

## 12. LOCKER_RENTAL TABLE

| Constraint Type | Column | Explanation |
|---|---|---|
| **PRIMARY KEY** | RENTAL_ID | Unique rental record. |
| **CHECK** | STATUS IN ('Active','Expired','Cancelled') | Correct rental statuses only. |
| **CHECK** | END_DATE > START_DATE | Prevents invalid rental durations. |
| **FOREIGN KEY** | LOCKER_ID → LOCKER | Rental must refer to a valid locker. |
| **FOREIGN KEY** | ACCOUNT_ID → ACCOUNT (CASCADE) | If account deleted → rental deleted. |

## Triggers, Procedures

## 1.1 Trigger: customer_bir

**Purpose:** Auto generates customer ID using sequence.

```
375
376  -- Auto-increment trigger
377  CREATE OR REPLACE TRIGGER customer_bir
378      BEFORE INSERT ON CUSTOMER FOR EACH ROW
379  BEGIN
380      IF :new.CUSTOMER_ID IS NULL THEN
381          SELECT customer_seq.NEXTVAL INTO :new.CUSTOMER_ID FROM dual;
382      END IF;
383  END;
384  /
385
```

## 2. USER AUTHENTICATION

## 2.1 Trigger: user_auth_bir

**Purpose:** Gives each new user an automatic USER_ID.

```
386  CREATE OR REPLACE TRIGGER user_auth_bir
387      BEFORE INSERT ON USER_AUTH FOR EACH ROW
388  BEGIN
389      IF :new.USER_ID IS NULL THEN
390          SELECT user_auth_seq.NEXTVAL INTO :new.USER_ID FROM dual;
391      END IF;
392  END;
393  /
394
```

## 2.2 Procedure: CREATE_USER

**Purpose:** Creates a complete user in one step → Customer + UserAuth + Account + Card.

This procedure registers a new user automatically with all required records.

```
CREATE OR REPLACE PROCEDURE CREATE_USER (
    p_full_name       IN VARCHAR2,
    p_email           IN VARCHAR2,
    p_password        IN VARCHAR2,
    p_cnic            IN VARCHAR2 DEFAULT NULL,
    p_phone           IN VARCHAR2 DEFAULT NULL,
    p_address         IN VARCHAR2 DEFAULT NULL,
    p_dob             IN DATE DEFAULT NULL,
    p_card_number     IN VARCHAR2 DEFAULT NULL,
    p_initial_balance IN NUMBER DEFAULT 50000,
    p_user_id         OUT NUMBER
)
AS
    v_customer_id   NUMBER;
    v_user_id       NUMBER;
    v_account_id    NUMBER;
    v_branch_id     NUMBER;
    v_type_id       NUMBER;
    v_first_name    VARCHAR2(50);
    v_last_name     VARCHAR2(50);

    -- Local variables (we can assign to these)
    v_cnic          VARCHAR2(20);
    v_phone         VARCHAR2(20);
    v_address       VARCHAR2(200);
    v_dob           DATE;
    v_card          VARCHAR2(30);
BEGIN
    ----------------------------------------------------
    -- Name split
    ----------------------------------------------------
    v_first_name := REGEXP_SUBSTR(p_full_name, '^\S+');
    v_last_name  := NVL(REGEXP_SUBSTR(p_full_name, '\s(.+)$', 1, 1), '');

    ----------------------------------------------------
    -- Default CNIC
```

## 3. ACCOUNT DOMAIN

### 3.1 Trigger: account_bir

**Purpose:** Auto generates ACCOUNT_ID before insert.

```
394
395 CREATE OR REPLACE TRIGGER branch_bir
396     BEFORE INSERT ON BRANCH FOR EACH ROW
397 BEGIN
398     IF :new.BRANCH_ID IS NULL THEN
399         SELECT branch_seq.NEXTVAL INTO :new.BRANCH_ID FROM dual;
400     END IF;
401 END;
402 /
```

## 3.2 Trigger: generate_account_number

**Purpose:** Creates an account number like *ACC50001* automatically.

```
495
496 CREATE OR REPLACE TRIGGER generate_account_number
497     BEFORE INSERT ON ACCOUNT
498     FOR EACH ROW
499 BEGIN
500     IF :new.ACCOUNT_NUMBER IS NULL THEN
501         SELECT 'ACC' || TO_CHAR(account_seq.NEXTVAL)
502         INTO :new.ACCOUNT_NUMBER FROM dual;
503     END IF;
504 END;
505 /
```

## 3.3 Procedure: UPDATE_BALANCE

**Purpose:** Sets account balance to a new value.

```
833
834 CREATE OR REPLACE PROCEDURE UPDATE_BALANCE (
835     p_user_id IN NUMBER,
836     p_new_balance IN NUMBER
837 )
838 AS
839 BEGIN
840     UPDATE ACCOUNT a
841     SET a.BALANCE = p_new_balance
842     WHERE a.ACCOUNT_ID = (
843         SELECT ah.ACCOUNT_ID
844         FROM ACCOUNT_HOLDER ah
845         JOIN USER_AUTH ua
846             ON ah.CUSTOMER_ID = ua.CUSTOMER_ID
847         WHERE ua.USER_ID = p_user_id AND ah.HOLDER_TYPE='Primary'
848     );
849 END;
```

## 3.4 Procedure: ADD_BALANCE

**Purpose:** Adds money to user's account.

```
/
CREATE OR REPLACE PROCEDURE ADD_BALANCE (
    p_user_id IN NUMBER,
    p_amount IN NUMBER
)
AS
BEGIN
    UPDATE ACCOUNT a
    SET a.BALANCE = a.BALANCE + p_amount
    WHERE a.ACCOUNT_ID = (
        SELECT ah.ACCOUNT_ID
        FROM ACCOUNT_HOLDER ah
        JOIN USER_AUTH ua ON ah.CUSTOMER_ID = ua.CUSTOMER_ID
        WHERE ua.USER_ID = p_user_id AND ah.HOLDER_TYPE='Primary'
    );
END;
/
```

## 3.5 Procedure: SUB_BALANCE

**Purpose:** Subtracts money from user's account.

```
/
CREATE OR REPLACE PROCEDURE SUB_BALANCE (
    p_user_id IN NUMBER,
    p_amount IN NUMBER
)
AS
BEGIN
    UPDATE ACCOUNT a
    SET a.BALANCE = a.BALANCE - p_amount
    WHERE a.ACCOUNT_ID = (
        SELECT ah.ACCOUNT_ID
        FROM ACCOUNT_HOLDER ah
        JOIN USER_AUTH ua ON ah.CUSTOMER_ID = ua.CUSTOMER_ID
        WHERE ua.USER_ID = p_user_id AND ah.HOLDER_TYPE='Primary'
    );
END;
```

# 4. CARD DOMAIN

## 4.1 Trigger: card_bir

**Purpose:** Auto assigns CARD_ID using sequence.

```
/

CREATE OR REPLACE TRIGGER card_bir
    BEFORE INSERT ON CARD FOR EACH ROW
BEGIN
    IF :new.CARD_ID IS NULL THEN
        SELECT card_seq.NEXTVAL INTO :new.CARD_ID FROM dual;
    END IF;
END;
/
```

## 4.2 Trigger: generate_card_number

**Purpose:** Creates a 16 digit card number automatically.

```
/

CREATE OR REPLACE TRIGGER generate_card_number
    BEFORE INSERT ON CARD
    FOR EACH ROW
BEGIN
    IF :new.CARD_NUMBER IS NULL THEN
        SELECT LPAD(card_seq.NEXTVAL, 16, '0')
        INTO :new.CARD_NUMBER FROM dual;
    END IF;
END;
/
```

# 5. TRANSACTION DOMAIN

## 5.1 Trigger: transaction_bir

**Purpose:** Auto assigns TRANSACTION_ID.

```
/

CREATE OR REPLACE TRIGGER transaction_bir
    BEFORE INSERT ON BANK_TRANSACTION FOR EACH ROW
BEGIN
    IF :new.TRANSACTION_ID IS NULL THEN
        SELECT transaction_seq.NEXTVAL INTO :new.TRANSACTION_ID FROM dual;
    END IF;
END;
/
```

## 5.2 Procedure: CREATE_TRANSACTION

**Purpose:** Handles fund transfer or deposit + validates name + updates balance.

This procedure saves every transaction and adjusts money safely.

```
882  CREATE OR REPLACE PROCEDURE CREATE_TRANSACTION (
883      p_user_id            IN NUMBER,
884      p_recipient_name     IN VARCHAR2,
885      p_recipient_card     IN VARCHAR2,
886      p_amount             IN NUMBER,
887      p_type               IN VARCHAR2,
888      p_transaction_id     OUT NUMBER
889  )
890  AS
891      v_sender_acc       NUMBER;
892      v_rec_acc          NUMBER;
893      v_balance          NUMBER;
894      v_clean_card       VARCHAR2(30);
895      v_db_name          VARCHAR2(200);
896      v_norm_db_name     VARCHAR2(200);
897      v_norm_input_name  VARCHAR2(200);
898  BEGIN
899      -- Clean card number
900      v_clean_card := REGEXP_REPLACE(p_recipient_card, '[^0-9]', '');
901      ------------------------------------------------------
902      -- Get sender account
903      ------------------------------------------------------
904      SELECT a.ACCOUNT_ID, a.BALANCE
905      INTO v_sender_acc, v_balance
906      FROM ACCOUNT a
907      JOIN ACCOUNT_HOLDER ah ON a.ACCOUNT_ID = ah.ACCOUNT_ID
908      JOIN USER_AUTH ua ON ah.CUSTOMER_ID = ua.CUSTOMER_ID
909      WHERE ua.USER_ID = p_user_id
910        AND ah.HOLDER_TYPE = 'Primary';
911      ------------------------------------------------------
912      -- If DEPOSIT, sender = receiver
913      ------------------------------------------------------
914      IF UPPER(p_type) = 'DEPOSIT' THEN
915          v_rec_acc := v_sender_acc;
916
917      ELSE
```

# 6. LOAN DOMAIN

## 6.1 Trigger: loan_type_bir

**Purpose:** Auto generates LOAN_TYPE_ID.

```
438  /
439
440  CREATE OR REPLACE TRIGGER loan_type_bir
441      BEFORE INSERT ON LOAN_TYPE FOR EACH ROW
442  BEGIN
443      IF :new.LOAN_TYPE_ID IS NULL THEN
444          SELECT loan_type_seq.NEXTVAL INTO :new.LOAN_TYPE_ID FROM dual;
445      END IF;
446  END;
447  /
```

## 6.2 Trigger: loan_application_bir

**Purpose:** Auto generates APPLICATION_ID.

```
448
449  CREATE OR REPLACE TRIGGER loan_application_bir
450      BEFORE INSERT ON LOAN_APPLICATION FOR EACH ROW
451  BEGIN
452      IF :new.APPLICATION_ID IS NULL THEN
453          SELECT loan_application_seq.NEXTVAL INTO :new.APPLICATION_ID FROM dual;
454      END IF;
455  END;
```

## 6.3 Trigger: loan_account_bir

**Purpose:** Auto generates LOAN_ACCOUNT_ID.

```
457
458  CREATE OR REPLACE TRIGGER loan_account_bir
459      BEFORE INSERT ON LOAN_ACCOUNT FOR EACH ROW
460  BEGIN
461      IF :new.LOAN_ACCOUNT_ID IS NULL THEN
462          SELECT loan_account_seq.NEXTVAL INTO :new.LOAN_ACCOUNT_ID FROM dual;
463      END IF;
464  END;
```

## 6.4 Procedure: APPLY_LOAN

**Purpose:** Inserts a new loan application for a customer.
Customer applies for a loan through this procedure.

```
656  CREATE OR REPLACE PROCEDURE APPLY_LOAN(
657      p_customer_id INT,
658      p_branch_id   INT,
659      p_type_id     INT,
660      p_amount      DECIMAL
661  ) AS
662  BEGIN
663      INSERT INTO LOAN_APPLICATION (CUSTOMER_ID, BRANCH_ID, LOAN_TYPE_ID, REQUESTED_AMOUNT)
664      VALUES (p_customer_id, p_branch_id, p_type_id, p_amount);
665  END;
```

# 7. LOCKER MANAGEMENT DOMAIN

## 7.1 Trigger: locker_bir

**Purpose:** Auto assigns LOCKER_ID.

```
468
469  CREATE OR REPLACE TRIGGER locker_bir
470      BEFORE INSERT ON LOCKER FOR EACH ROW
471  BEGIN
472      IF :new.LOCKER_ID IS NULL THEN
473          SELECT locker_seq.NEXTVAL INTO :new.LOCKER_ID FROM dual;
474      END IF;
475  END;
```

## 7.2 Trigger: locker_rental_bir

**Purpose:** Auto assigns RENTAL_ID.

```
478  CREATE OR REPLACE TRIGGER locker_rental_bir
479      BEFORE INSERT ON LOCKER_RENTAL FOR EACH ROW
480  BEGIN
481      IF :new.RENTAL_ID IS NULL THEN
482          SELECT locker_rental_seq.NEXTVAL INTO :new.RENTAL_ID FROM dual;
483      END IF;
484  END;
485  /
```

## 7.3 Trigger: locker_set_occupied

**Purpose:** When rental is created → locker becomes Occupied.
If someone books a locker, it is marked occupied.

```
632  CREATE OR REPLACE TRIGGER locker_set_occupied
633      AFTER INSERT ON LOCKER_RENTAL
634      FOR EACH ROW
635  BEGIN
636      UPDATE LOCKER
637      SET STATUS = 'Occupied'
638      WHERE LOCKER_ID = :new.LOCKER_ID;
639  END;
```

## 7.4 Trigger: locker_set_available

**Purpose:** When rental expires or is cancelled , locker becomes Available.
(If rental ends, locker frees automatically.)

```sql
CREATE OR REPLACE TRIGGER locker_set_available
    AFTER UPDATE OF STATUS ON LOCKER_RENTAL
    FOR EACH ROW
BEGIN
    IF :new.STATUS IN ('Expired', 'Cancelled') THEN
        UPDATE LOCKER
        SET STATUS = 'Available'
        WHERE LOCKER_ID = :new.LOCKER_ID;
    END IF;
END;
```

## 7.5 Procedure: RENT_LOCKER

**Purpose:** Creates a new locker rental record.
(This procedure assigns a locker to a user.)

```sql
CREATE OR REPLACE PROCEDURE RENT_LOCKER(
    p_locker_id  INT,
    p_account_id INT,
    p_end_date   DATE
) AS
BEGIN
    INSERT INTO LOCKER_RENTAL (LOCKER_ID, ACCOUNT_ID, END_DATE)
    VALUES (p_locker_id, p_account_id, p_end_date);
END;
/

COMMIT;
```

# 8. LOCKER AUTO EXPIRY & CLEANUP SYSTEM

## 8.1 Table: EXPIRED_RENTAL_QUEUE

**Purpose:** Temporarily stores expired rental IDs for cleanup.

```sql

CREATE TABLE EXPIRED_RENTAL_QUEUE (
    RENTAL_ID NUMBER PRIMARY KEY
);
```

## 8.2 Trigger: trg_collect_expired_rentals

**Purpose:** When a rental becomes Expired , add Rental_ID to queue.
(Marks expired rentals for cleaning.)

```
1008  CREATE OR REPLACE TRIGGER trg_collect_expired_rentals
1009  AFTER UPDATE OF STATUS ON LOCKER_RENTAL
1010  FOR EACH ROW
1011  WHEN (NEW.STATUS = 'Expired')
1012  BEGIN
1013      INSERT INTO EXPIRED_RENTAL_QUEUE (RENTAL_ID)
1014      VALUES (:NEW.RENTAL_ID);
1015  END;
```

## 8.3 Trigger: trg_cleanup_expired_rentals

**Purpose:**

- Frees locker
- Deletes expired rental
- Clears queue

(Cleans expired rentals automatically.)

```
1017  CREATE OR REPLACE TRIGGER trg_cleanup_expired_rentals
1018  AFTER UPDATE ON LOCKER_RENTAL
1019  BEGIN
1020      -- Make the locker available again
1021      UPDATE LOCKER
1022      SET STATUS = 'Available'
1023      WHERE LOCKER_ID IN (
1024          SELECT LOCKER_ID
1025          FROM LOCKER_RENTAL
1026          WHERE RENTAL_ID IN (SELECT RENTAL_ID FROM EXPIRED_RENTAL_QUEUE)
1027      );
1028
1029      -- Delete expired rentals
1030      DELETE FROM LOCKER_RENTAL
1031      WHERE RENTAL_ID IN (SELECT RENTAL_ID FROM EXPIRED_RENTAL_QUEUE);
1032
1033      -- Clear temp queue
1034      DELETE FROM EXPIRED_RENTAL_QUEUE;
1035  END;
```

## 9. SEQUENCE TRIGGERS (AUTO ID GENERATORS)

These all work the same way : **assign IDs from sequences**:

| Trigger Name | Table | Purpose |
|---|---|---|
| branch_bir | BRANCH | New branch ID |
| account_type_bir | ACCOUNT_TYPE | New account type |
| locker_bir | LOCKER | New locker ID |
| locker_rental_bir | LOCKER_RENTAL | New rental ID |
| loan_type_bir | LOAN_TYPE | New loan type ID |
| loan_application_bir | LOAN_APPLICATION | New application ID |

| | | |
|---|---|---|
| loan_account_bir | LOAN_ACCOUNT | New loan account ID |
| transaction_bir | BANK_TRANSACTION | New transaction ID |
| customer_bir | CUSTOMER | New customer ID |
| user_auth_bir | USER_AUTH | New user ID |

These triggers help Oracle auto fill IDs.

## Insertion of Test Data

## 1. CUSTOMER

**Purpose***:* To create basic customers for testing login, accounts, lockers, loans.

```
542  INSERT INTO CUSTOMER (CNIC, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, PHONE_NUMBER, ADDRESS, EMAIL)
543  VALUES ('1234567890123', 'Ahmed', 'Khan', TO_DATE('1990-01-15', 'YYYY-MM-DD'), '03001234567', 'House 1, Street 2, Karachi', 'ahmed.khan@example.
544
545  INSERT INTO CUSTOMER (CNIC, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, PHONE_NUMBER, ADDRESS, EMAIL)
546  VALUES ('2345678901234', 'Fatima', 'Ali', TO_DATE('1985-05-20', 'YYYY-MM-DD'), '03011234568', 'House 3, Street 4, Lahore', 'fatima.ali@example.c
547
548  INSERT INTO CUSTOMER (CNIC, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, PHONE_NUMBER, ADDRESS, EMAIL)
549  VALUES ('3456789012345', 'Usman', 'Malik', TO_DATE('1982-08-10', 'YYYY-MM-DD'), '03021234569', 'House 5, Street 6, Islamabad', 'usman.malik@exam
550
551  INSERT INTO CUSTOMER (CNIC, FIRST_NAME, LAST_NAME, DATE_OF_BIRTH, PHONE_NUMBER, ADDRESS, EMAIL)
552  VALUES ('4567890123456', 'Ayesha', 'Rehman', TO_DATE('1988-12-05', 'YYYY-MM-DD'), '03031234570', 'House 7, Street 8, Rawalpindi', 'ayesha.rehman(
```

## 2. USER AUTH

**Purpose***:* Enables login functionality for test users.

```
572  -- Insert User Auth for customers (using actual customer IDs that were generated)
573  INSERT INTO USER_AUTH (CUSTOMER_ID, PASSWORD_HASH, STATUS)
574  SELECT CUSTOMER_ID, '$2b$12$p/8tjQynfnXRPujl1wCzI.siIYrqdfMxsl0JqWtZnG9Ws8lJip14S', 'Active'
575  FROM CUSTOMER;
576
577  COMMIT;
```

## 3. BRANCH

**Purpose***:* Required for account creation and loan applications.

```
522  --  Branches
523  INSERT INTO BRANCH (BRANCH_NAME, LOCATION) VALUES ('Main Branch', 'City Center');
524  INSERT INTO BRANCH (BRANCH_NAME, LOCATION) VALUES ('North Branch', 'North Area');
525  INSERT INTO BRANCH (BRANCH_NAME, LOCATION) VALUES ('South Branch', 'South Area');
```

## 4. ACCOUNT TYPE

**Purpose***:* Needed to create accounts and validate minimum balance.

```
526
527  --  Account Types
528  INSERT INTO ACCOUNT_TYPE (TYPE_NAME, MIN_BALANCE, MONTHLY_FEE)
529  VALUES ('Savings Account', 1000, 0);
530  INSERT INTO ACCOUNT_TYPE (TYPE_NAME, MIN_BALANCE, MONTHLY_FEE)
531  VALUES ('Current Account', 5000, 100);
532  INSERT INTO ACCOUNT_TYPE (TYPE_NAME, MIN_BALANCE, MONTHLY_FEE)
533  VALUES ('Savings Account', 2000, 0);
```

## 5. ACCOUNT + ACCOUNT HOLDER

**Purpose***:* Links customers to their accounts for all testing.

```
-- ===== ACCOUNT TEST DATA =====
INSERT INTO ACCOUNT (BRANCH_ID, TYPE_ID, BALANCE, ACCOUNT_MODE, STATUS)
VALUES (1, 1, 50000, 'Individual', 'Active');

INSERT INTO ACCOUNT (BRANCH_ID, TYPE_ID, BALANCE, ACCOUNT_MODE, STATUS)
VALUES (2, 2, 250000, 'Individual', 'Active');


-- ===== ACCOUNT_HOLDER TEST DATA =====
INSERT INTO ACCOUNT_HOLDER (ACCOUNT_ID, CUSTOMER_ID, HOLDER_TYPE)
VALUES (50001, 1001, 'Primary');

INSERT INTO ACCOUNT_HOLDER (ACCOUNT_ID, CUSTOMER_ID, HOLDER_TYPE)
VALUES (50002, 1002, 'Primary');
```

## 6. CARD

**Purpose***:* Enables transfer tests using cards.

```
INSERT INTO CARD (ACCOUNT_ID, CARD_NUMBER, CARD_TYPE, EXPIRY_DATE, STATUS, DAILY_LIMIT)
VALUES (50001, '5555444433332222', 'Debit', ADD_MONTHS(SYSDATE, 60), 'Active', 100000);

INSERT INTO CARD (ACCOUNT_ID, CARD_NUMBER, CARD_TYPE, EXPIRY_DATE, STATUS, DAILY_LIMIT)
VALUES (50002, '4444333322221111', 'Debit', ADD_MONTHS(SYSDATE, 60), 'Active', 150000);

```

## 7. TRANSACTION

**Purpose***:* Tests deposit + transfer + remaining balance updates.

```
INSERT INTO BANK_TRANSACTION (
    ACCOUNT_ID, RECIPIENT_ACCOUNT_ID, RECIPIENT_ACCOUNT_NAME,
    AMOUNT, TRANSACTION_TYPE, TRANSACTION_MODE, BALANCE_REMAINING
)
VALUES (50001, 50002, 'Fatima Ali', 2000, 'Transfer', 'Online', 48000);

```

## 8. LOAN

**Purpose***:* Allows you to test loan approval, triggers, closing logic.

```
INSERT INTO LOAN_TYPE (TYPE_NAME, PROFIT_RATE, MAX_DURATION_MONTHS, MIN_AMOUNT, MAX_AMOUNT)
VALUES ('Housing', 5.5, 360, 500000, 50000000);
INSERT INTO LOAN_TYPE (TYPE_NAME, PROFIT_RATE, MAX_DURATION_MONTHS, MIN_AMOUNT, MAX_AMOUNT)
VALUES ('Car', 7.2, 60, 500000, 5000000);

```

## 9. LOCKER

**Purpose***:* Basic lockers for testing unlock/lock triggers.

```sql
INSERT INTO LOAN_TYPE (TYPE_NAME, PROFIT_RATE, MAX_DURATION_MONTHS, MIN_AMOUNT, MAX_AMOUNT)
VALUES ('Housing', 5.5, 360, 500000, 50000000);
INSERT INTO LOAN_TYPE (TYPE_NAME, PROFIT_RATE, MAX_DURATION_MONTHS, MIN_AMOUNT, MAX_AMOUNT)
VALUES ('Car', 7.2, 60, 500000, 5000000);
```

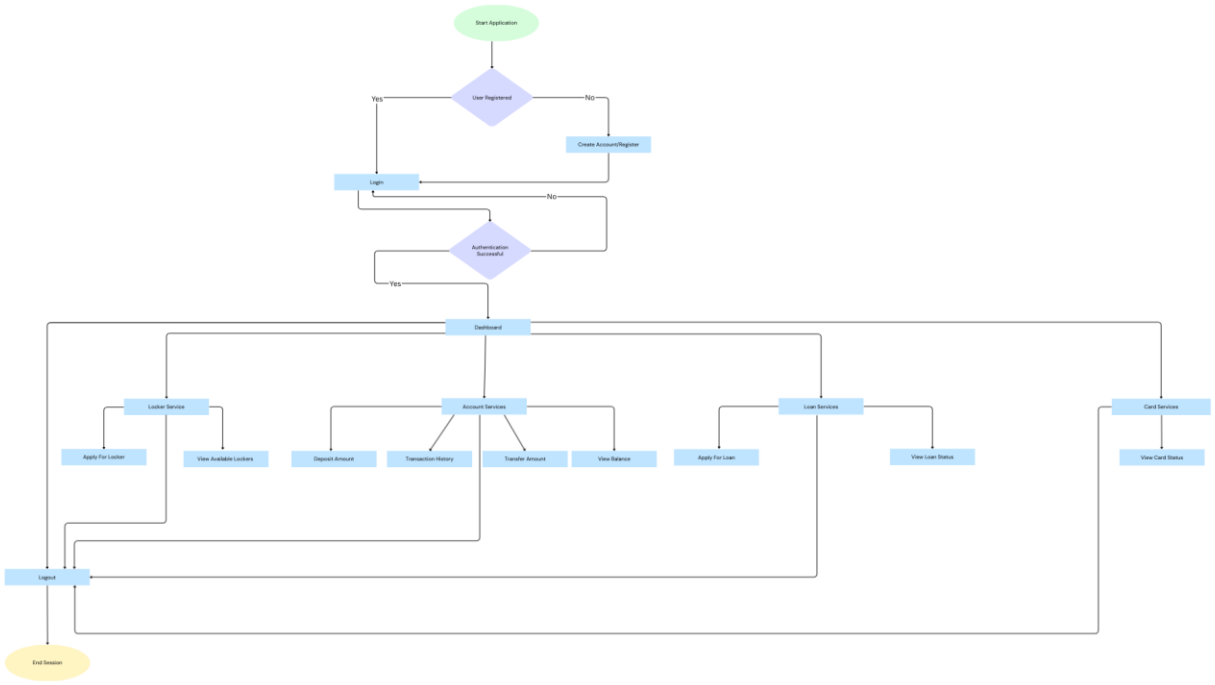
## 10. LOCKER RENTAL

**Purpose**: Tests locker triggers:

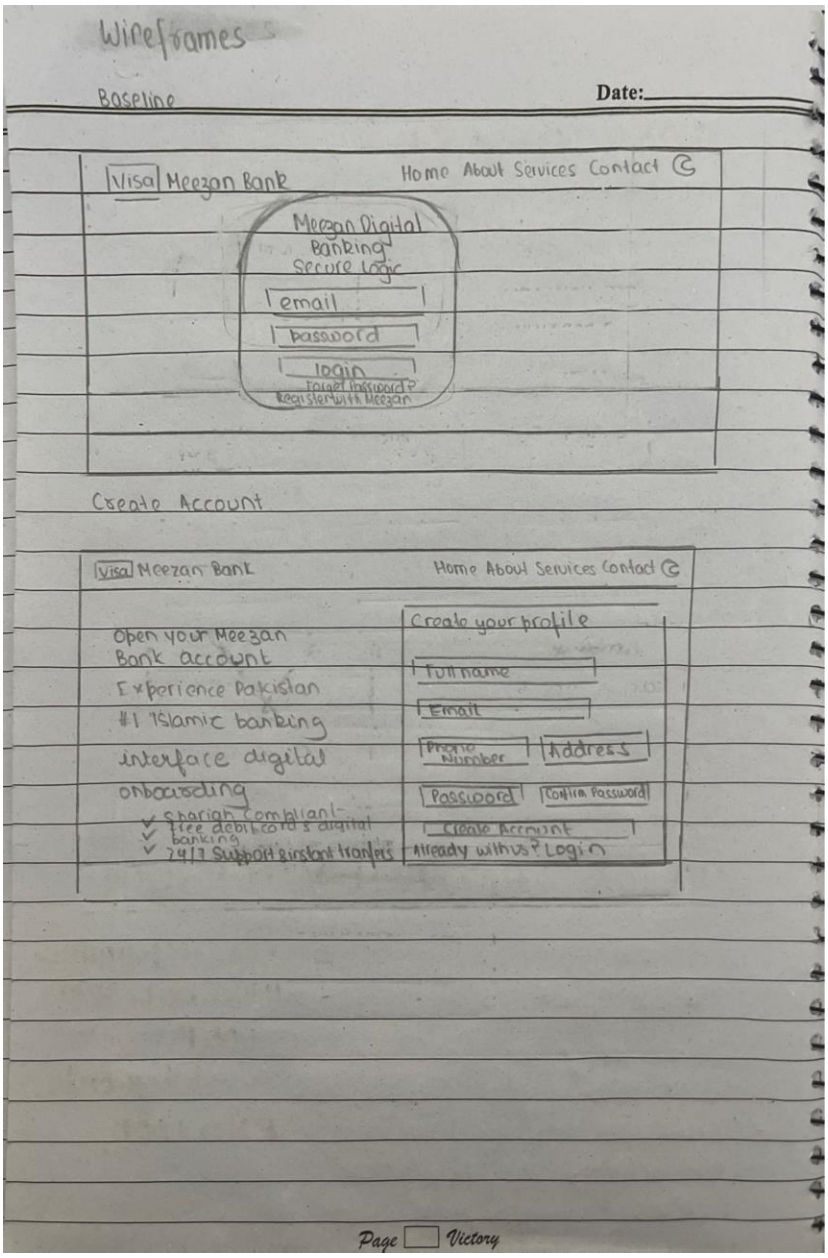
- locker_set_occupied
- locker_set_available
- expired locker cleanup

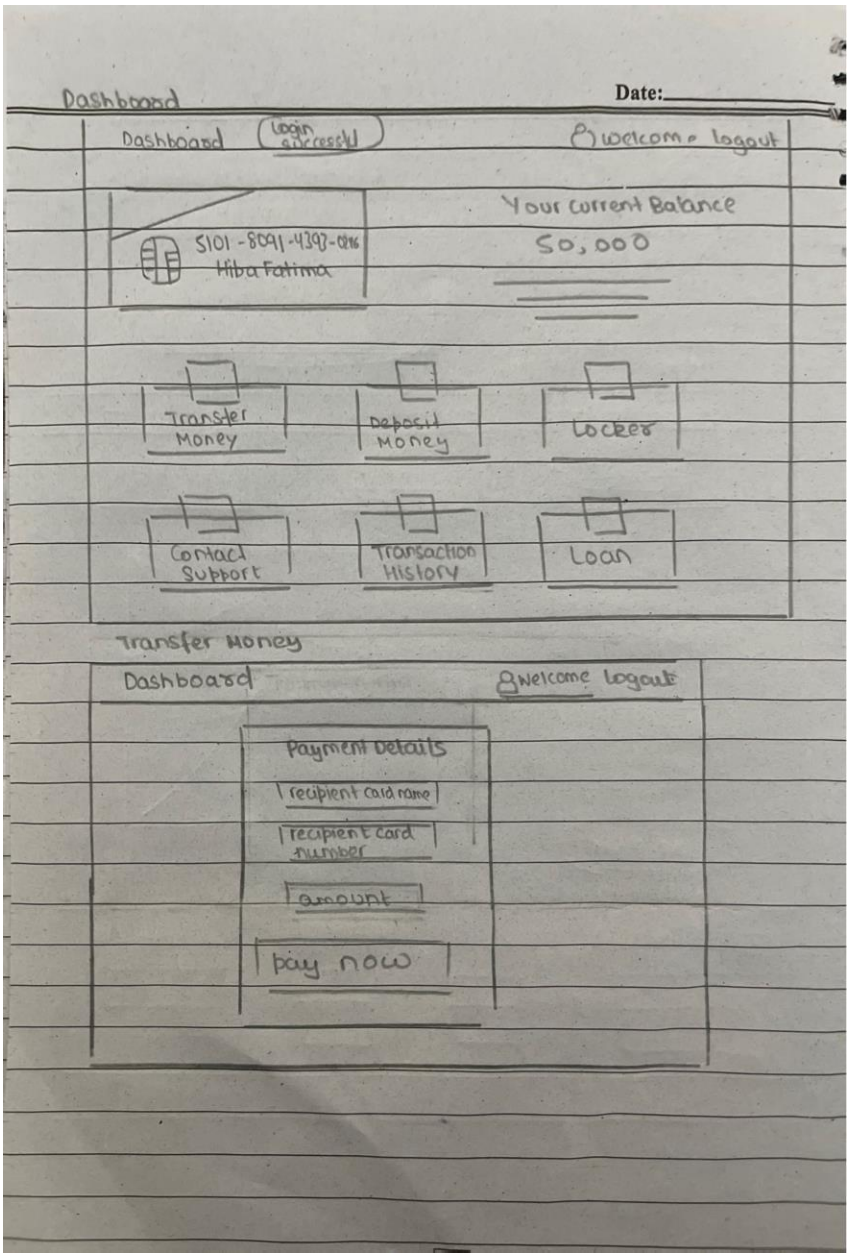
```sql
INSERT INTO LOCKER_RENTAL (LOCKER_ID, ACCOUNT_ID, END_DATE)
VALUES (1, 50001, DATE '2025-12-09');

INSERT INTO LOCKER_RENTAL (LOCKER_ID, ACCOUNT_ID, END_DATE)
VALUES (2, 50002, DATE '2025-12-09');
```

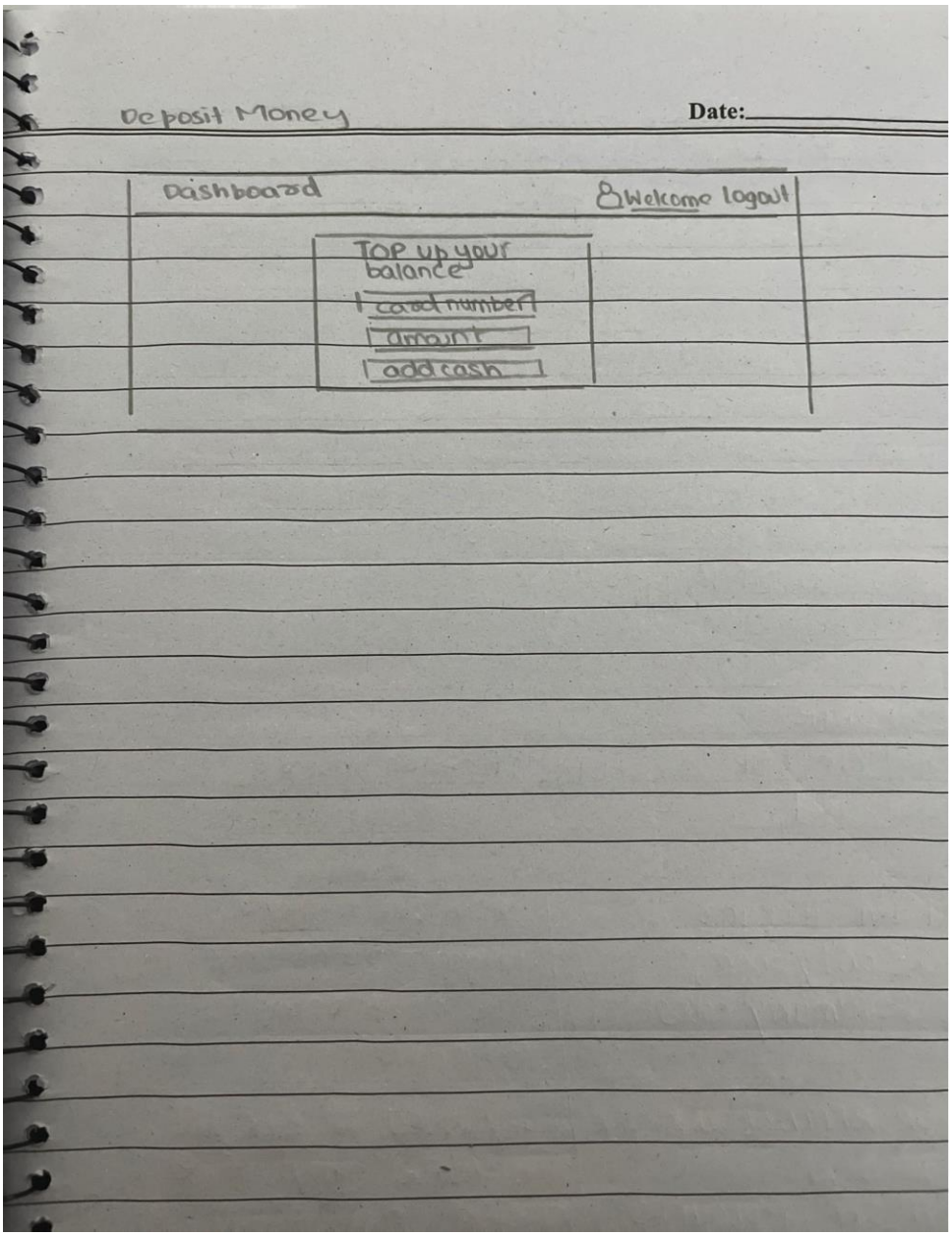
## Flow diagram

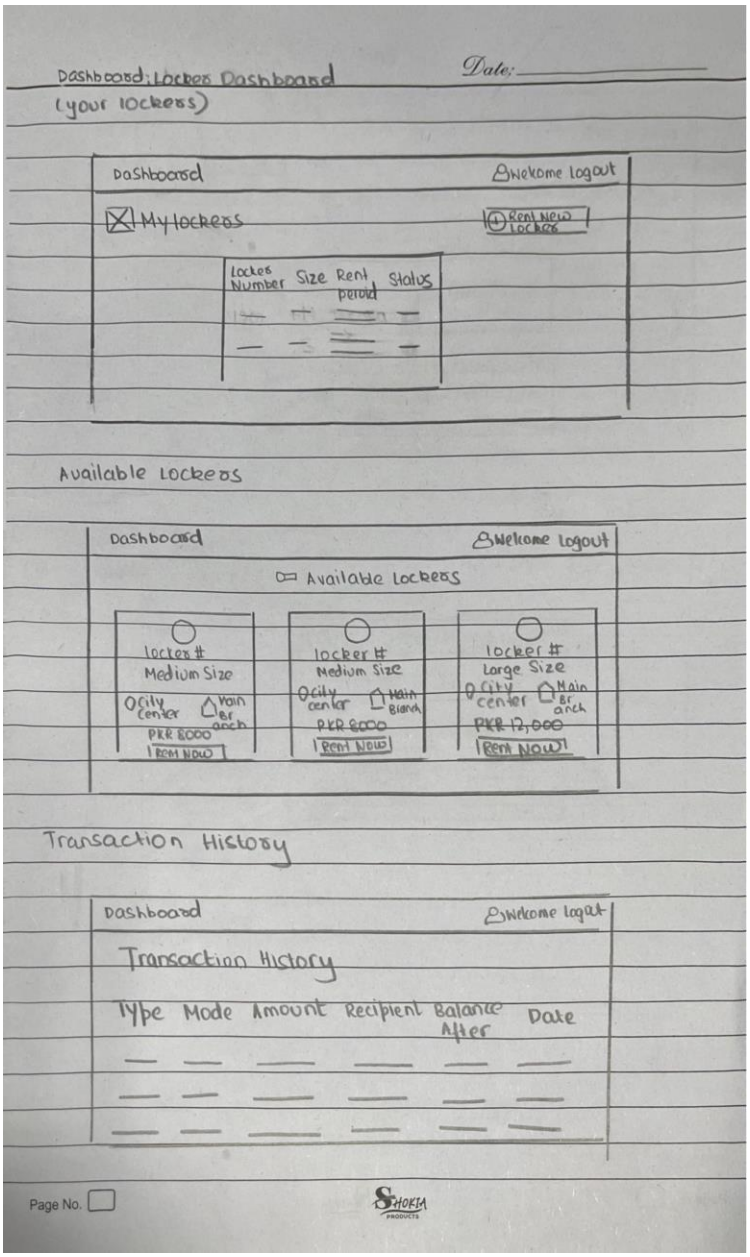


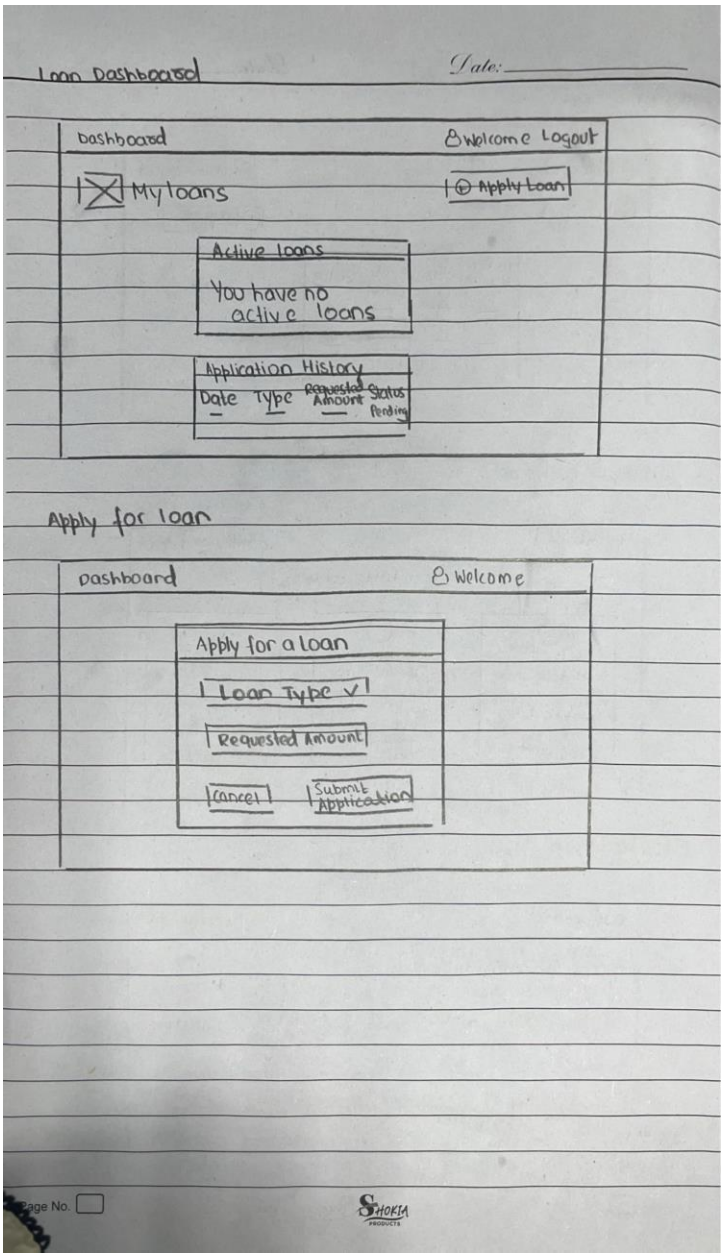

## Wireframes


Wireframes
Baseline
Date:
Visa Meezan Bank
Home About Services Contact
Meezan Digital Banking Secure Login
email
password
login
Forget Password?
Register with Meezan
Create Account
Visa Meezan Bank
Home About Services Contact
Open your Meezan Bank account
Experience Pakistan #1 Islamic banking interface digital onboarding
shariah compliant-
free debit card's digital banking
24/7 Support & instant transfers
Create your profile
Full name
Email
Phone Number
Address
Password
Confirm Password
Create Account
Already with us? Login
Page
Victory

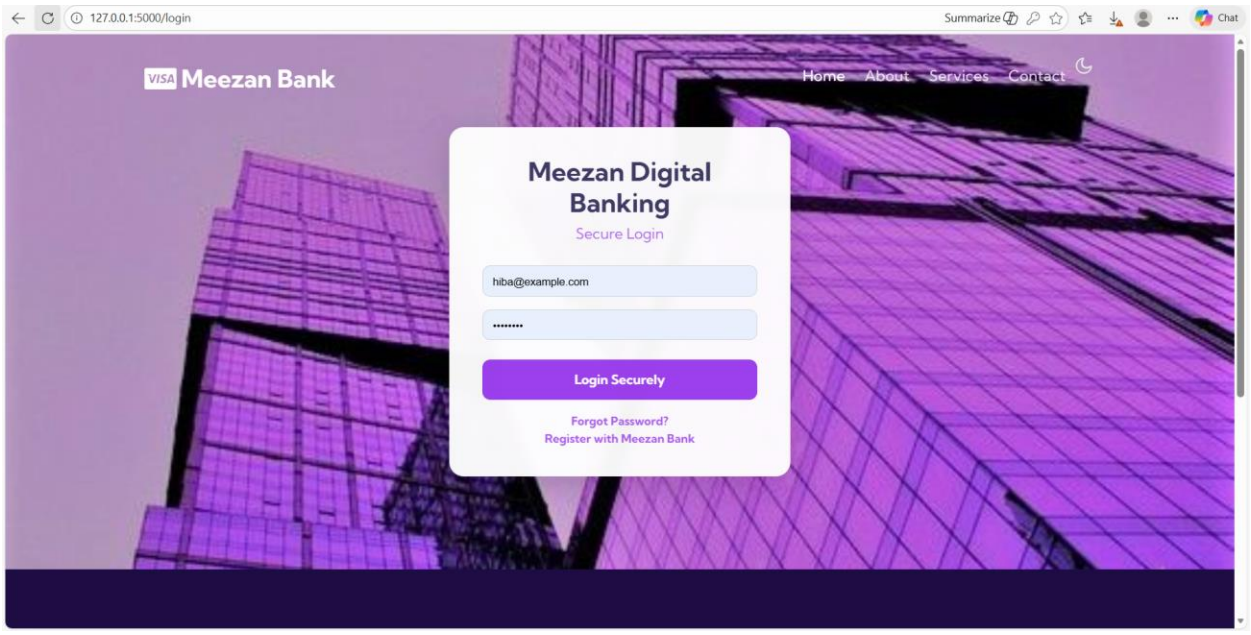

Dashboard
Date:
Dashboard
Login successfully
Welcome Logout
5101-8091-4393-0216
Hiba Fatima
Your current Balance
50,000
Transfer Money
Deposit Money
Locker
Contact Support
Transaction History
Loan
Transfer Money
Dashboard
Welcome Logout
Payment Details
recipient card name
recipient card number
amount
pay now


Deposit Money
Date:
Dashboard
Welcome logout
TOP up your balance
card number
amount
add cash

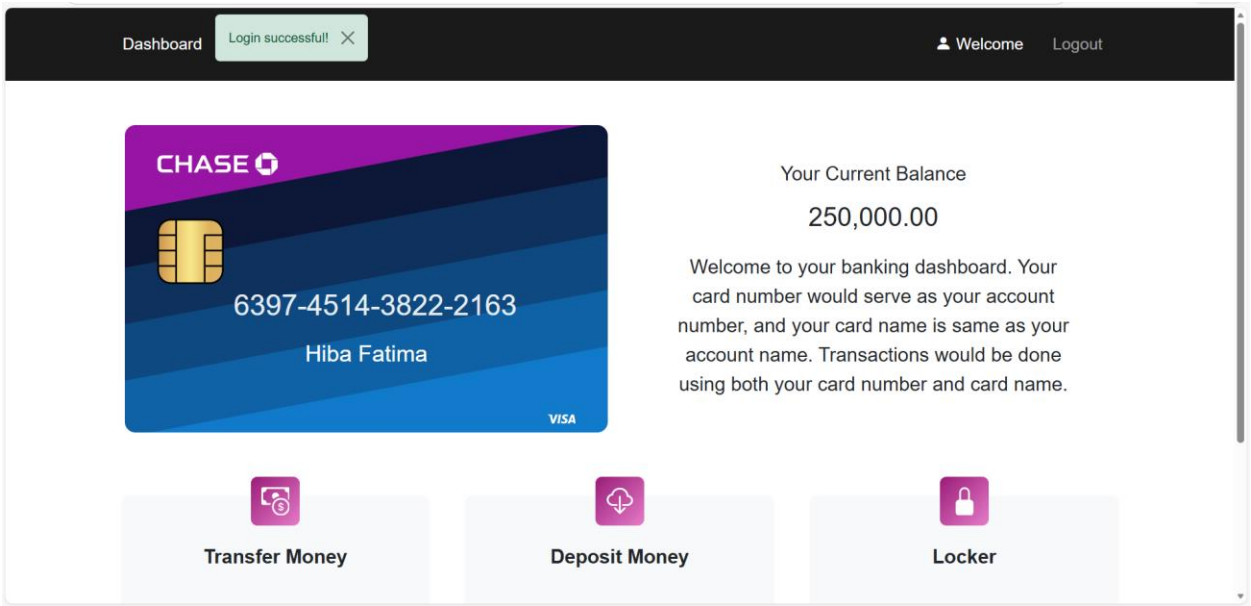

Dashboard: Lockers Dashboard
Date:
(your lockers)
Dashboard
Welcome Logout
My lockers
Rent New Locker
Locker Number
Size
Rent period
Status
Available Lockers
Dashboard
Welcome Logout
Available Lockers
locker #
Medium Size
City center
Main Branch
PKR 8000
Rent Now
locker #
Medium Size
City center
Main Branch
PKR 8000
Rent Now
locker #
Large Size
City center
Main Branch
PKR 12,000
Rent Now
Transaction History
Dashboard
Welcome Logout
Transaction History
Type
Mode
Amount
Recipient
Balance After
Date
Page No.
SHOKIA
PRODUCTS




## Page-by-Page Navigation and SQL Queries




## 1. DASHBOARD SCREEN




## Purpose of the Screen

The dashboard provides the user with a summary of their account, including:

- Current balance
- Card information
- Navigation to transactions History, deposits, transfers, loans, and locker pages

## SQL Query Used

When the user logs in, the backend runs a query to fetch account and card details:

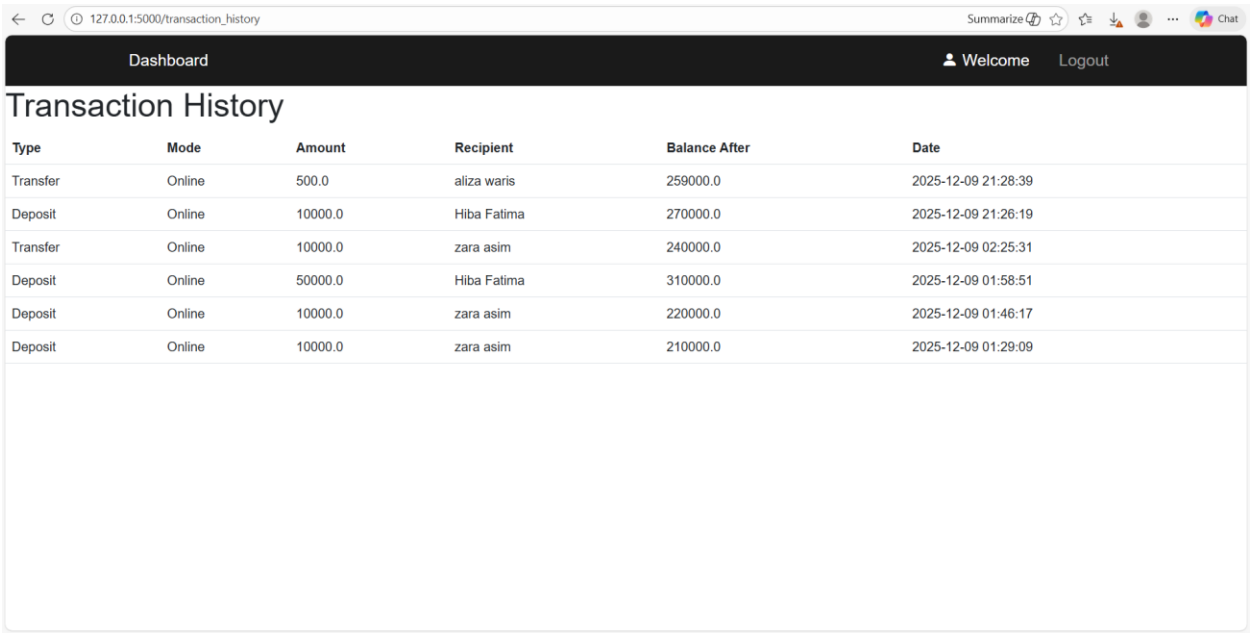
```
SELECT a.BALANCE, c.CARD_NUMBER, cust.FIRST_NAME || ' ' || cust.LAST_NAME AS full_name
FROM ACCOUNT a
JOIN ACCOUNT_HOLDER ah ON a.ACCOUNT_ID = ah.ACCOUNT_ID
JOIN USER_AUTH ua ON ah.CUSTOMER_ID = ua.CUSTOMER_ID
JOIN CUSTOMER cust ON cust.CUSTOMER_ID = ah.CUSTOMER_ID
JOIN CARD c ON c.ACCOUNT_ID = a.ACCOUNT_ID
WHERE ua.USER_ID = :logged_in_user_id;
```

## Explanation (Functionality Contribution)

- Fetches the user's **account balance** → displayed on dashboard
- Fetches **card number and cardholder name** → displayed on card image
- ensures the logged in user only sees **their own data**

This query powers the main card display and Your Current Balance area.

## 2. TRANSACTION HISTORY SCREEN




## Purpose of the Screen

Shows all transactions performed by the user:

- Transfer money
- Deposits
- Fees
- Withdrawals

## SQL Query Used

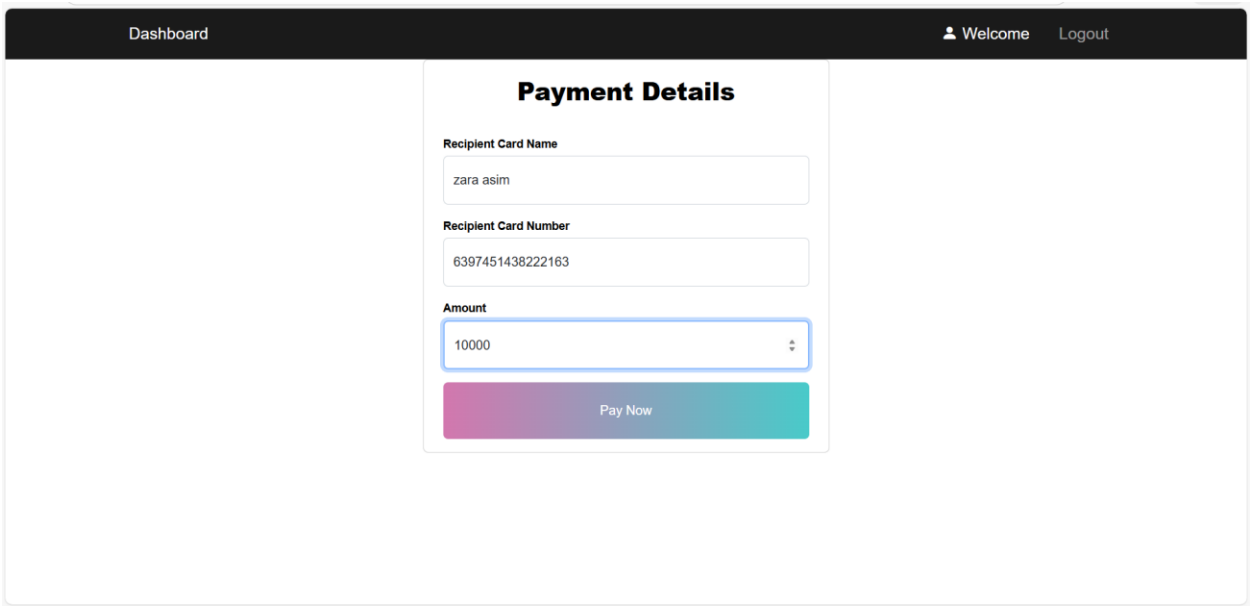
```
SELECT
    TRANSACTION_TYPE,
    TRANSACTION_MODE,
    AMOUNT,
    RECIPIENT_ACCOUNT_NAME,
    BALANCE_REMAINING,
    TRANSACTION_DATE
FROM BANK_TRANSACTION
WHERE ACCOUNT_ID = (
    SELECT ACCOUNT_ID
    FROM ACCOUNT_HOLDER ah
    JOIN USER_AUTH ua ON ah.CUSTOMER_ID = ua.CUSTOMER_ID
    WHERE ua.USER_ID = :user_id
)
ORDER BY TRANSACTION_DATE DESC;
```

## Explanation

- Identifies the current user's primary account
- Retrieves all transactions for that account
- Orders them by latest date
- Data shown in table columns:
    - Type: TRANSACTION_TYPE
    - Mode: TRANSACTION_MODE
    - Amount: AMOUNT
    - Recipient: RECIPIENT_ACCOUNT_NAME
    - Balance After: BALANCE_REMAINING
    - Date: TRANSACTION_DATE

This builds the **entire transaction history table** seen in the screenshot.

## 3. TRANSFER MONEY PAGE (Payment Details)



## Purpose

Allows users to:

- Enter recipient name
- Enter recipient card number
- Enter amount
- Perform a transfer

## SQL Query Used Inside CREATE_TRANSACTION Procedure

```
-- Get sender account
-------------------------------------------------------
SELECT a.ACCOUNT_ID, a.BALANCE
INTO v_sender_acc, v_balance
FROM ACCOUNT a
JOIN ACCOUNT_HOLDER ah ON a.ACCOUNT_ID = ah.ACCOUNT_ID
JOIN USER_AUTH ua ON ah.CUSTOMER_ID = ua.CUSTOMER_ID
WHERE ua.USER_ID = p_user_id
  AND ah.HOLDER_TYPE = 'Primary';
```

This identifies the sender's account balance.

Then:

```
-------------------------------------------------------
-- Get recipient details
-------------------------------------------------------
SELECT c.ACCOUNT_ID,
       cust.FIRST_NAME || ' ' || cust.LAST_NAME
INTO v_rec_acc, v_db_name
FROM CARD c
JOIN ACCOUNT_HOLDER ah ON c.ACCOUNT_ID = ah.ACCOUNT_ID
JOIN CUSTOMER cust ON cust.CUSTOMER_ID = ah.CUSTOMER_ID
WHERE REGEXP_REPLACE(c.CARD_NUMBER, '[^0-9]', '') = v_clean_card
  AND c.STATUS = 'Active';
```

This Identifies the correct recipient using card number.

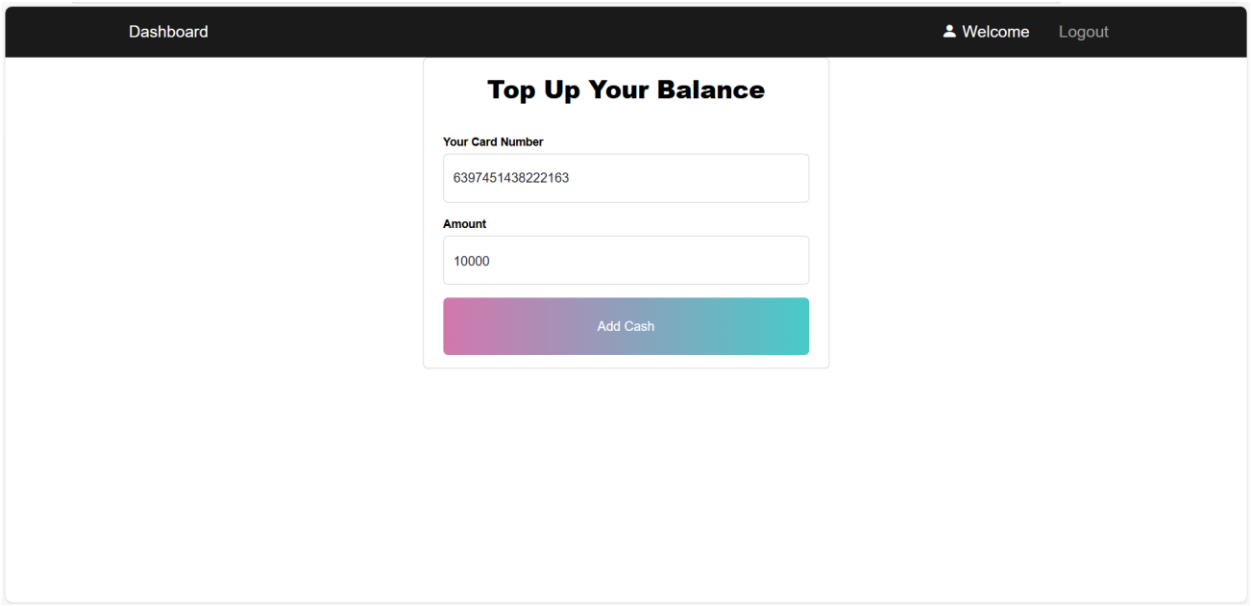
Finally, inserts transaction:

```
    INSERT INTO BANK_TRANSACTION (
        ACCOUNT_ID,
        RECIPIENT_ACCOUNT_ID,
        RECIPIENT_ACCOUNT_NAME,
        AMOUNT,
        TRANSACTION_TYPE,
        TRANSACTION_MODE,
        BALANCE_REMAINING
    )
    VALUES (
        v_sender_acc,
        v_rec_acc,
        p_recipient_name,
        p_amount,
        p_type,
        'Online',
        CASE
            WHEN UPPER(p_type) = 'DEPOSIT' THEN v_balance + p_amount
            ELSE v_balance - p_amount
        END
    )
    RETURNING TRANSACTION_ID INTO p_transaction_id;
    COMMIT;
END;
```

## Explanation (Functionality)

- Validates card number and recipient name
- Checks user balance
- Deducts amount for Transfer
- Inserts new transaction row
- Updates remaining balance

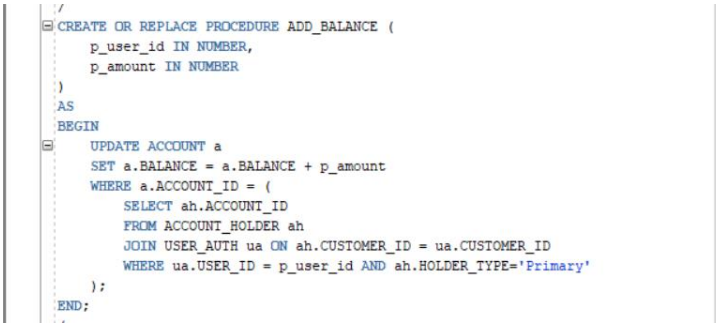
## 4. DEPOSIT MONEY PAGE (Top Up Balance)



## Purpose

User adds money to their account.

## SQL Query Used (Procedure ADD_BALANCE)



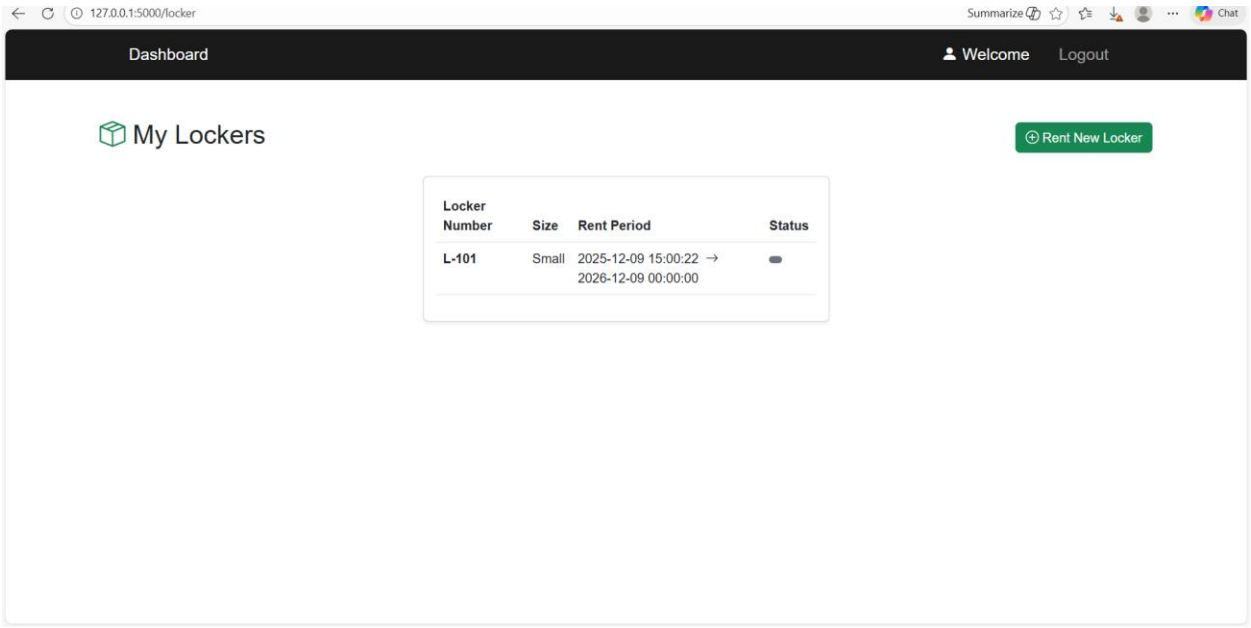
```
/
CREATE OR REPLACE PROCEDURE ADD_BALANCE (
    p_user_id IN NUMBER,
    p_amount IN NUMBER
)
AS
BEGIN
    UPDATE ACCOUNT a
    SET a.BALANCE = a.BALANCE + p_amount
    WHERE a.ACCOUNT_ID = (
        SELECT ah.ACCOUNT_ID
        FROM ACCOUNT_HOLDER ah
        JOIN USER_AUTH ua ON ah.CUSTOMER_ID = ua.CUSTOMER_ID
        WHERE ua.USER_ID = p_user_id AND ah.HOLDER_TYPE='Primary'
    );
END;
```

## Explanation (Functionality)

- Finds logged in user's account
- Adds the deposit amount
- Updates balance on dashboard
- Displays success message

This is what makes the “Top Up Your Balance” form actually update the account.

## 5. MY LOCKERS PAGE




## Purpose

Shows all lockers currently rented by the user.

## SQL Query Used in backend

```
SELECT
    l.LOCKER_NUMBER,
    l.LOCKER_SIZE,
    lr.START_DATE,
    lr.END_DATE,
    lr.STATUS
FROM LOCKER_RENTAL lr
JOIN LOCKER l ON lr.LOCKER_ID = l.LOCKER_ID
JOIN ACCOUNT a ON lr.ACCOUNT_ID = a.ACCOUNT_ID
JOIN ACCOUNT_HOLDER ah ON a.ACCOUNT_ID = ah.ACCOUNT_ID
JOIN USER_AUTH ua ON ah.CUSTOMER_ID = ua.CUSTOMER_ID
WHERE ua.USER_ID = :user_id;
```

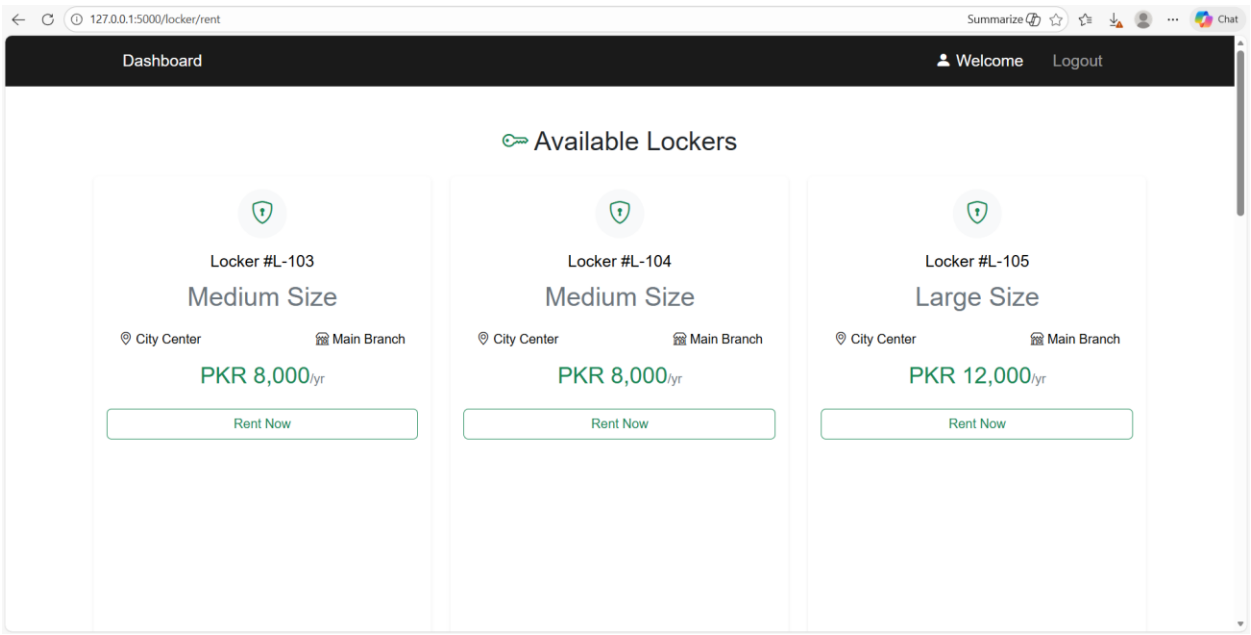
## Explanation

- Identifies all lockers where the user's account is renting
- Displays:
    - Locker number
    - Size

- Rent period
- Status

This builds the **My Lockers card** shown in screenshot.

## 6. AVAILABLE LOCKERS PAGE (Rent New Locker)




## Purpose

Shows only lockers that are:

- Available
- Not currently rented
- Belong to any branch

## SQL Query Used

```sql
SELECT
    LOCKER_ID,
    LOCKER_NUMBER,
    LOCKER_SIZE,
    ANNUAL_FEE,
    BRANCH_ID
FROM LOCKER
WHERE STATUS = 'Available';
```

```sql
INSERT INTO LOCKER (BRANCH_ID, LOCKER_NUMBER, LOCKER_SIZE, ANNUAL_FEE, STATUS) VALUES (1, 'L-101', 'Small', 5000, 'Available');
INSERT INTO LOCKER (BRANCH_ID, LOCKER_NUMBER, LOCKER_SIZE, ANNUAL_FEE, STATUS) VALUES (1, 'L-102', 'Small', 5000, 'Available');
INSERT INTO LOCKER (BRANCH_ID, LOCKER_NUMBER, LOCKER_SIZE, ANNUAL_FEE, STATUS) VALUES (1, 'L-103', 'Medium', 8000, 'Available');
INSERT INTO LOCKER (BRANCH_ID, LOCKER_NUMBER, LOCKER_SIZE, ANNUAL_FEE, STATUS) VALUES (1, 'L-104', 'Medium', 8000, 'Available');
INSERT INTO LOCKER (BRANCH_ID, LOCKER_NUMBER, LOCKER_SIZE, ANNUAL_FEE, STATUS) VALUES (1, 'L-105', 'Large', 12000, 'Available');

-- Branch 2 (North Branch)
INSERT INTO LOCKER (BRANCH_ID, LOCKER_NUMBER, LOCKER_SIZE, ANNUAL_FEE, STATUS) VALUES (2, 'L-201', 'Small', 5000, 'Available');
INSERT INTO LOCKER (BRANCH_ID, LOCKER_NUMBER, LOCKER_SIZE, ANNUAL_FEE, STATUS) VALUES (2, 'L-202', 'Medium', 8000, 'Available');

-- Branch 3 (South Branch)
INSERT INTO LOCKER (BRANCH_ID, LOCKER_NUMBER, LOCKER_SIZE, ANNUAL_FEE, STATUS) VALUES (3, 'L-301', 'Small', 5000, 'Available');
INSERT INTO LOCKER (BRANCH_ID, LOCKER_NUMBER, LOCKER_SIZE, ANNUAL_FEE, STATUS) VALUES (3, 'L-302', 'Large', 12000, 'Available');

COMMIT;
```

## Explanation

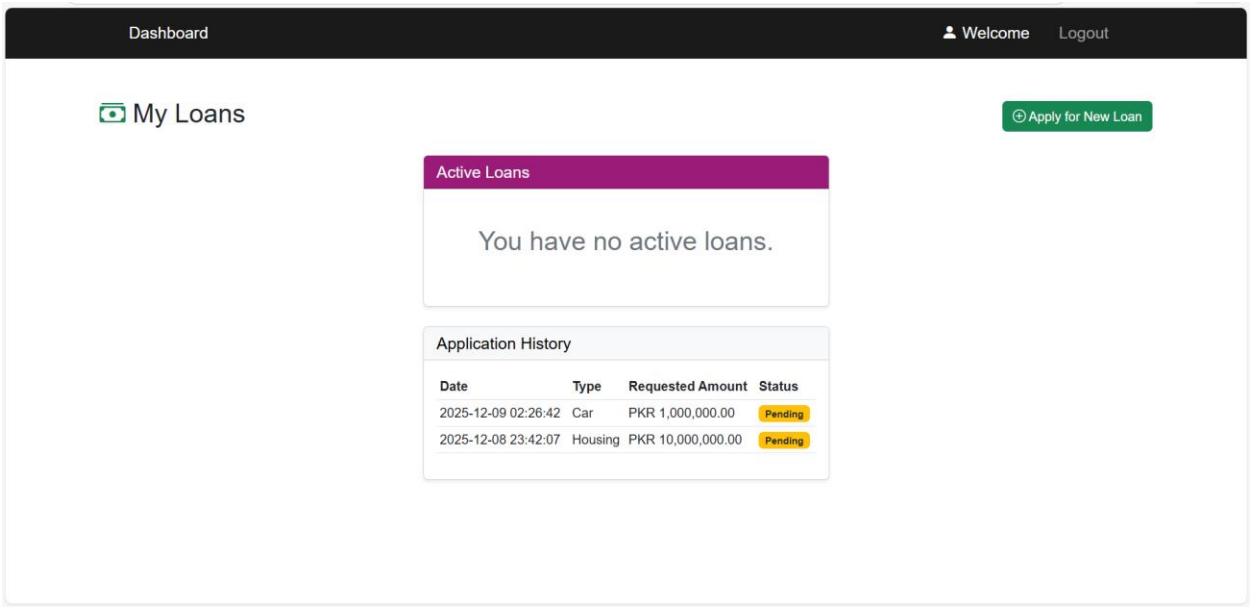
- Fetches all lockers that are not occupied
- Used to populate the cards:
    - Medium Size
    - PKR 8,000/yr
    - “Rent Now” button

When user rents, another SQL executes:

## Rent Locker Procedure:

```sql
CREATE OR REPLACE PROCEDURE RENT_LOCKER(
    p_locker_id  INT,
    p_account_id INT,
    p_end_date   DATE
) AS
BEGIN
    INSERT INTO LOCKER_RENTAL (LOCKER_ID, ACCOUNT_ID, END_DATE)
    VALUES (p_locker_id, p_account_id, p_end_date);
END;
/

COMMIT;
```

## 7. LOAN OVERVIEW PAGE




**Purpose:**

Allows customers to submit loan requests and view loan application history. Loan processing is handled by a separate department.

## SQL Query : Loan Application History

```
SELECT
    la.APPLICATION_DATE,
    lt.TYPE_NAME,
    la.REQUESTED_AMOUNT,
    la.STATUS
FROM LOAN_APPLICATION la
JOIN LOAN_TYPE lt ON la.LOAN_TYPE_ID = lt.LOAN_TYPE_ID
WHERE la.CUSTOMER_ID = :user_id
ORDER BY la.APPLICATION_DATE DESC;
```

**Explanation:**

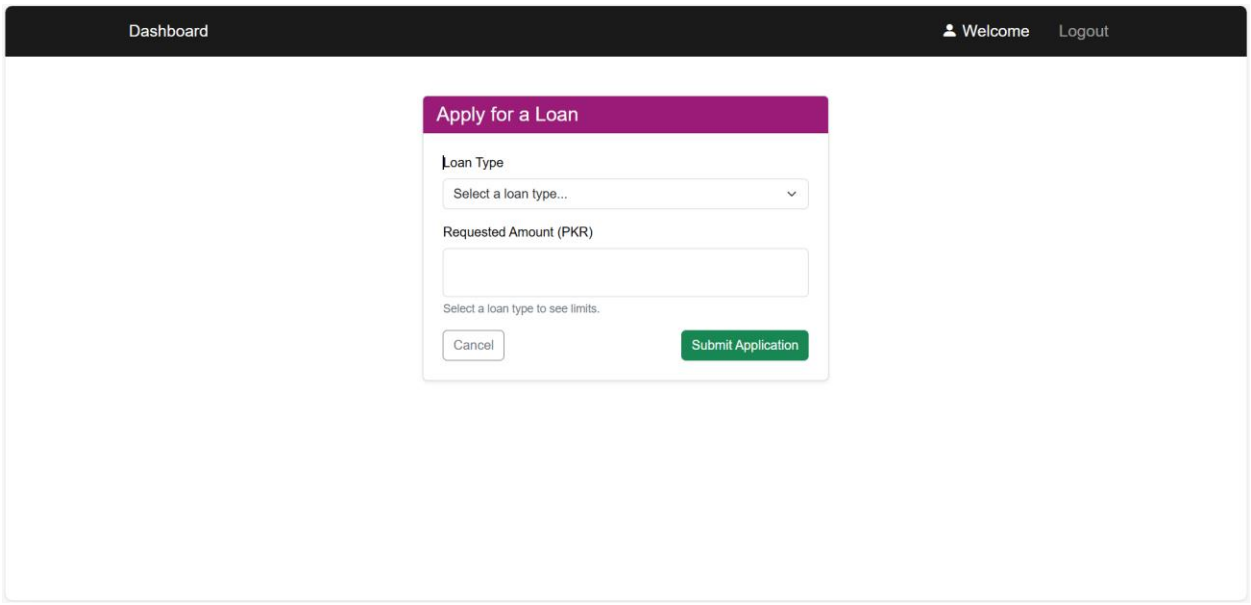
This query lists all loan requests submitted by the customer and displays:

- Application Date
- Loan Type

- Requested Amount
- Current Status

Used to show the history table in the Loan Overview screen.

## 8. APPLY FOR LOAN PAGE




## Purpose

User applies for a new car or housing loan.

## SQL Query Used : APPLY_LOAN Procedure

```
-- 5. APPLY_LOAN Procedure
CREATE OR REPLACE PROCEDURE APPLY_LOAN(
    p_customer_id INT,
    p_branch_id   INT,
    p_type_id     INT,
    p_amount      DECIMAL
) AS
BEGIN
    INSERT INTO LOAN_APPLICATION (CUSTOMER_ID, BRANCH_ID, LOAN_TYPE_ID, REQUESTED_AMOUNT)
    VALUES (p_customer_id, p_branch_id, p_type_id, p_amount);
END;
```

### Explanation

- Creates a new loan request
- Default STATUS = 'Pending'
- Appears in Application History table
- Can later be updated by admin