BankSight: Transaction Intelligence Dashboard

User Manual

1. Overview

BankSight is a Streamlit web application that builds a mini banking analytics platform on top of a SQLite database. It uses six core datasets:

- `customers_clean.csv`
- `accounts_clean.csv`
- `transactions_clean.csv`
- `branches.csv`
- `loans.csv`
- `support_tickets.csv`

The app loads these CSV files into `sbi_bank.db` at startup and provides:

- Interactive table browsing
- Multi-level filtering on any table
- Full CRUD (Create, Read, Update, Delete) operations
- Credit / Debit simulation with a minimum balance rule
- 15+ analytical SQL insights (customers, transactions, loans, branches, support tickets)
- Basic fraud/anomaly-style queries

The app is implemented in `app.py` (or `sbi_dashboard.py`) and is designed to run both locally and on Streamlit Cloud.

---

2. Installation and Setup

2.1. Requirements

- Python 3.9+
- Pip
- Git (for cloning from GitHub)

Python libraries (installed via `requirements.txt`):

- `streamlit`
- `pandas`
- `numpy`
- `sqlite3` (standard library)

2.2. Folder Structure

Place the following files in the same directory:

- `app.py`
- `customers_clean.csv`
- `accounts_clean.csv`
- `transactions_clean.csv`
- `branches.csv`
- `loans.csv`
- `support_tickets.csv`
- `requirements.txt`

The app will create or overwrite `sbi_bank.db` in this folder on startup.

2.3. Installation Steps

1. Clone the repository:

   ```
   bash
   git clone https://github.com/<your-username>/<your-repo-name>.git
   cd <your-repo-name>
   ```

2. (Optional but recommended) Create and activate a virtual environment.
3. Install dependencies:

   ```
   bash
   pip install -r requirements.txt
   ```

---

3. Running the Application

From the project folder, run:

```
bash
streamlit run app.py
```

or:

```
bash
python -m streamlit run app.py
```

Streamlit will open the app in your browser (usually at `http://localhost:8501`).

On startup, the app:

- Connects to `sbi_bank.db`
- Drops any existing copies of the six tables
- Loads the six CSVs into fresh tables in SQLite
- Enables foreign keys

---

4. Data Model

The SQLite database `sbi_bank.db` contains six main tables:

- `customers`
  - `customer_id` (primary identifier)
  - Demographics: `name`, `gender`, `age`, `city`, `account_type`, `join_date`
- `accounts`
  - `customer_id`
  - `account_balance`
  - `last_updated`
- `transactions`
  - `txn_id`
  - `customer_id`
  - `txn_type` (deposit, withdrawal, transfer, purchase, online fraud, etc.)
  - `amount`
  - `txn_time`
  - `status` (success / failed)
- `branches`
  - `Branch_ID`
  - `Branch_Name`
  - `City`
  - `Manager_Name`
  - `Total_Employees`, `Branch_Revenue`, `Opening_Date`, `Performance_Rating`
- `loans`
  - `Loan_ID`
  - `Customer_ID`
  - `Account_ID`
  - `Branch`
  - `Loan_Type`, `Loan_Amount`, `Interest_Rate`, `Loan_Term_Months`
  - `Start_Date`, `End_Date`, `Loan_Status`
- `support_tickets`
  - `Ticket_ID`
  - `Customer_ID`
  - `Account_ID`
  - `Loan_ID` (can be blank)
  - `Branch_Name`
  - `Issue_Category`, `Description`
  - `Date_Opened`, `Date_Closed`
  - `Priority`, `Status`, `Resolution_Remarks`, `Support_Agent`, `Channel`
  - `Customer_Rating`

The app assumes consistent IDs across tables (e.g., `customers.customer_id` ↔ `accounts.customer_id` ↔ `transactions.customer_id`).

---

5. Navigation and Features

The left sidebar contains the main navigation radio:

- 🏠 Introduction
- 📊 View Tables
- 🔍 Filter Data
- 🖍️ CRUD Operations

- 💰 Credit / Debit Simulation
- 🧠 Analytical Insights
- 🧑‍💻 About Creator

5.1. 🏠 Introduction

- Describes the purpose of the dashboard.
- Lists datasets used and summarizes key features.
- Good place for a brief project overview.

5.2. 📊 View Tables

- Select any table from the dropdown.
- The app shows:
  - Row/column counts
  - Full table contents in a scrollable grid
- There is a **Refresh Table** button to reload from SQLite if anything changed via CRUD or simulations.

Index in the display starts from 1 for readability; the underlying data is unchanged.

5.3. 🔍 Filter Data

This page provides multi-level filtering on any dataset directly via SQL.

Steps:

1. Choose a table from the dropdown.
2. The app loads and shows all columns.
3. Select one or more columns in the “Columns to filter” multiselect.
4. For each selected column, choose a filter type:
   - `equals` – exact match
   - `contains` – substring (uses `LIKE '%value%'`)
   - `range` – between two values (useful for numeric or date text fields)
5. Click **Apply Filters** to:
   - Build a `SELECT ... WHERE ...` query
   - Display the SQL used
   - Show the filtered result table

If you do not click Apply, the unfiltered base table is shown.

5.4. 🖊 CRUD Operations

This page supports Create, Read, Update, and Delete for *any* table.

Layout:

- Left column:
  - Dropdown to select the target table
  - Radio to select action: `Create`, `Read`, `Update`, `Delete`
- Right column: context-specific UI for the chosen action.

General rules:

- The first column of the table is treated as the primary key (`pk_col`).
- Non-date inputs are usually converted to uppercase before writing (for consistency).
- Date/time fields like `last_updated`, `join_date`, `txn_time`, `Date_Opened`, `Date_Closed` allow blanks -> stored as `NULL`.

After each operation:

- The app commits changes to SQLite.
- It sets a `crud_updated` flag so other pages can refresh.
- It shows the affected row(s) in a result panel.

Details:

Create

- Shows a text input for every column.
- On “Insert”:
  - Runs an `INSERT INTO ... VALUES (...)` statement.
  - Displays the row that was just inserted (by PK).

Read

- Input: primary key value (optional).
- If PK is provided:
  - Returns only rows with that key.
- If left empty:
  - Returns the entire table.

Update

- Input:
  - Primary key to update
  - Column to update (dropdown)
  - New value
- On “Update”:
  - Executes an `UPDATE` query.
  - Shows the updated row.

Delete

- Input: primary key to delete.
- On “Delete”:
  - Executes `DELETE FROM ... WHERE pk_col = ?`.
  - Shows any remaining rows with that key (usually empty).

5.5. 💰 Credit / Debit Simulation

Simulates simple account operations with a minimum balance rule.

Workflow:

1. Enter a **Customer ID** (must exist in `accounts`).
2. The app fetches `account_balance` and displays it using a metric.
3. Choose operation:
   - `Deposit`
   - `Withdraw`
4. Enter the amount (must be > 0).
5. Click **Submit**:
   - For **Deposit**:
     - New balance = current + amount.
     - Updates `accounts.account_balance` and `last_updated`.
   - For **Withdraw**:
     - If `current_balance - amount < 1000`, the app shows an error and does not update.
     - Otherwise:
       - New balance = current - amount.
       - Updates `account_balance` and `last_updated`.
6. The result panel shows:
   - Type of transaction
   - Amount
   - New balance
   - Timestamp

This page intentionally writes to the same `accounts` table used everywhere else.

5.6. 🧠 Analytical Insights

This page presents 15+ pre-defined SQL insights. Each insight is represented as:

- A human-readable question in a dropdown.
- A corresponding SQL query stored in a dictionary.

When you:

1. Select a question from the dropdown.

2. Click **Run**.

The app:

- Displays the SQL text used.
- Executes the query live against `sbi_bank.db`.
- Shows the result table.

Example types of questions:

- Top 10 customers by total account balance
- Customers with 2023 account openings and balance > ₹100,000
- Transaction volume by transaction type
- Customers with >3 failed transactions in a month
- Top 5 branches by transaction volume (last 6 months)
- Average loan amount and interest rate by loan type
- Customers with multiple active/approved loans
- Branch performance summary (customers, loans, transaction volume)
- Issue categories with longest average resolution time
- Support agents handling the most critical tickets with high ratings
- Fraud-style patterns, such as customers with many failed or online fraud attempts

You can easily extend this section by adding more entries to the `questions` dictionary in the code.

5.7. 🧑‍💻 About Creator

- Static page describing the creator (bio, skills, interests, contact info).
- This text is fully editable inside the app's code.
- Intended for project evaluation / portfolio context.

---

6. Typical Usage Flow

1. **Start the app** (database is built and CSVs are loaded).
2. **Check data** in 📊 View Tables to verify everything looks correct.
3. **Use** 🔍 **Filter Data** to explore specific segments, e.g.,:
    - High-balance customers in a given city
    - Transactions of a given type and status
4. **Manipulate data** with ✏️ CRUD Operations:
    - Add a new synthetic customer or loan
    - Correct a test value
    - Delete test records
5. **Simulate real operations** in 💰 Credit / Debit Simulation:
    - Show how deposits and withdrawals update account balances live.
6. **Run insights** in 🧠 Analytical Insights for:
    - Banking KPIs
    - Risk and fraud-like patterns
    - Branch and loan performance

7. **Present the project** starting from 🏠 Introduction and 🧑‍💻 About Creator.

---

7. Deployment Notes

To deploy on Streamlit Community Cloud:

1. Push this project (including `app.py`, CSVs, and `requirements.txt`) to a GitHub repo.
2. Go to https://streamlit.io/cloud and create a new app.
3. Select your repo, branch, and set `app.py` as the main file.
4. Deploy; Streamlit will install dependencies and run the app in the cloud.

---

8. Customization

You can customize:

- **Branding:**
  - Sidebar logo and title (`show_logo` helper, if used).
  - Theme (via `.streamlit/config.toml` or Streamlit settings).
- **Questions and queries:**
  - Add, remove, or modify entries in the `questions` dictionary.
  - Create new analytical views focusing on fraud, performance, or customer segmentation.
- **CRUD behaviour:**
  - Add type-aware widgets (e.g., date pickers, number inputs) for specific columns.
  - Add validation rules (e.g., non-negative amounts, valid statuses).

---

9. Limitations

- Intended for small to medium datasets (fits well into in-memory SQLite + Pandas).
- Concurrency is basic: not designed for heavy multi-user production use.
- No authentication; anyone with the app URL can access it (in typical Streamlit Cloud deployments).

---

10. License and Attribution

- This project is for educational and portfolio purposes.
- Datasets are synthetic and meant only for demo/learning.
- If you reuse the structure, please credit the original author and adapt queries to your own data.