FINANCE FRAUD & LOANS DATASET

TABLE OF CONTENTS

INTRODUCTION

Financial fraud has become a growing concern in the banking sector, especially with the rise of digital transactions and online loan processing.

The **Finance Fraud & Loans Dataset** enables analysis of borrower details, credit history, income levels, and loan performance to identify patterns linked to fraudulent activities. Using data analytics, this project aims to detect anomalies, predict potential fraud, and improve the accuracy of loan risk assessments.

By uncovering key indicators of fraud, financial institutions can strengthen their decision-making processes, enhance security measures, and build greater trust and transparency in financial operations. A list of finance datasets with 50k+ transactions, customers, accounts, loans and reference tables is to be looked at for this project.

LOAD DATA TABLES

About the data tables

Tables include:

- **Core Entities**

customers.csv → Customer master data (id, type, demographics).

accounts.csv → Bank accounts linked to customers, with type & status.

loans.csv → Loans issued, with type, status, and customer references.

transactions.csv → Financial transactions with customer, account, branch, and type references.

- **Dimensions & Reference Tables**

customer_types.csv → Classification of customers (e.g. personal, corporate).

account_types.csv → Account categories (e.g. savings, checking).

account_statuses.csv → Account lifecycle states (e.g. active, closed).

loan_statuses.csv → Loan states (e.g. current, default).

transaction_types.csv → Transaction categories (e.g. payment, transfer, withdrawal).

branches.csv → Bank branch metadata (id, name, location).

addresses.csv → Address data linked to customers and branches.

Migration of Datasets into MySQL Workbench

Steps

1. Open MySQL Work bench.
2. Click on MYSQL Connections.
3. Click on the Create a new schema in the connected server icon.
4. Type the name of the Schema in the field provided on the schema dialogue box.
5. Click on Apply.
6. Click on Apply.
7. Click on Finish.
8. Click on the arrow beside the schema created.
9. Right click on the Table option.

10. From the dialogue box that appears, click on Table Data Import Wizard.

11. From the Select File to Import box, click on Browse to select the csv file to be migrated into MySQL work-bench.

12. Click on Next.

13. Check Drop table, if exists.

14. Click on Next.

15. Click on Next again and again.

16. Click on Finish.

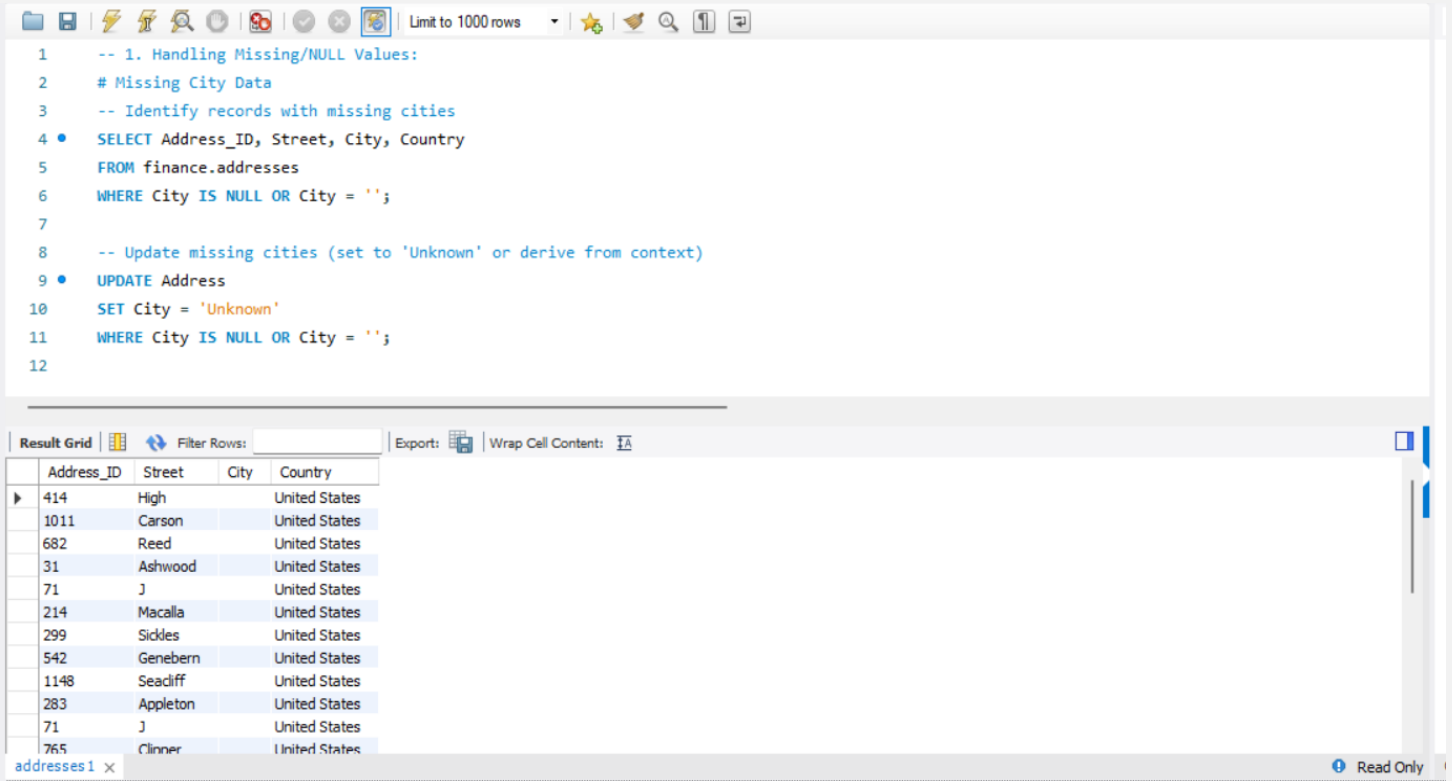
17. Click on Refresh.

BASIC CHECKS

Based on the provided CSV data tables, several data quality issues are identified, that require cleaning. Here are the various data cleaning techniques with MySQL syntaxes:

1. **Handling Missing/NULL Values:**

a. Missing City Data



b. Missing First Names

```
22
23 •  SELECT Customer_ID, First_Name, Last_Name
24    FROM finance.customers
25    WHERE First_Name IS NULL OR TRIM(First_Name) = '';
26
27
28
29
30
31
32
33
34
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Customer_ID	First_Name	Last_Name
10983		Battle
10830		Burt
10161		Pena
10145		Rogers
10179		James
11087		Cortez
10432		Morton
10890		Warner
10604		Boyle
10534		Wiggins
10669		Keith
10333		Rush

2. Standardizing Text Data

a. Standardizing Country Names

```sql
-- Check for inconsistent country entries
SELECT DISTINCT Country
FROM finance.addresses;

SET SQL_SAFE_UPDATES = 0;

UPDATE finance.addresses
SET Country = 'United States'
WHERE Country IN ('US', 'USA', 'United States', '');

SET SQL_SAFE_UPDATES = 1;    -- optional: turn back on
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Country
United States
United StateR
United Slates
United vtates
Unitd States
United Staes
UnitedcStates
United State
United StXtes
United0States
Pnited States

b. Standardizing Case in Names

```sql
--Standardizing Case in Names
SELECT
    Customer_ID,
    First_Name          AS Old_First_Name,
    Last_Name           AS Old_Last_Name,

    CONCAT(
        UPPER(SUBSTRING(First_Name, 1, 1)),
        LOWER(SUBSTRING(First_Name, 2, LENGTH(First_Name) - 1))
    )                   AS New_First_Name,

    CONCAT(
        UPPER(SUBSTRING(Last_Name, 1, 1)),
        LOWER(SUBSTRING(Last_Name, 2, LENGTH(Last_Name) - 1))
    )                   AS New_Last_Name

FROM finance.customers
WHERE Customer_ID IS NOT NULL
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Customer_ID	Old_First_Name	Old_Last_Name	New_First_Name	New_Last_Name
10000	Maybell	Acevedo	Maybell	Acevedo
10001	Sherrell	Barlow	Sherrell	Barlow
10003	Marcelo	Everett	Marcelo	Everett
10004	Natashia	Gillespie	Natashia	Gillespie
10005	Orlando	Valencia	Orlando	Valencia
10006	Fumiko	Case	Fumiko	Case

3. Handling Invalid Dates

b. Fixing Invalid Date Formats

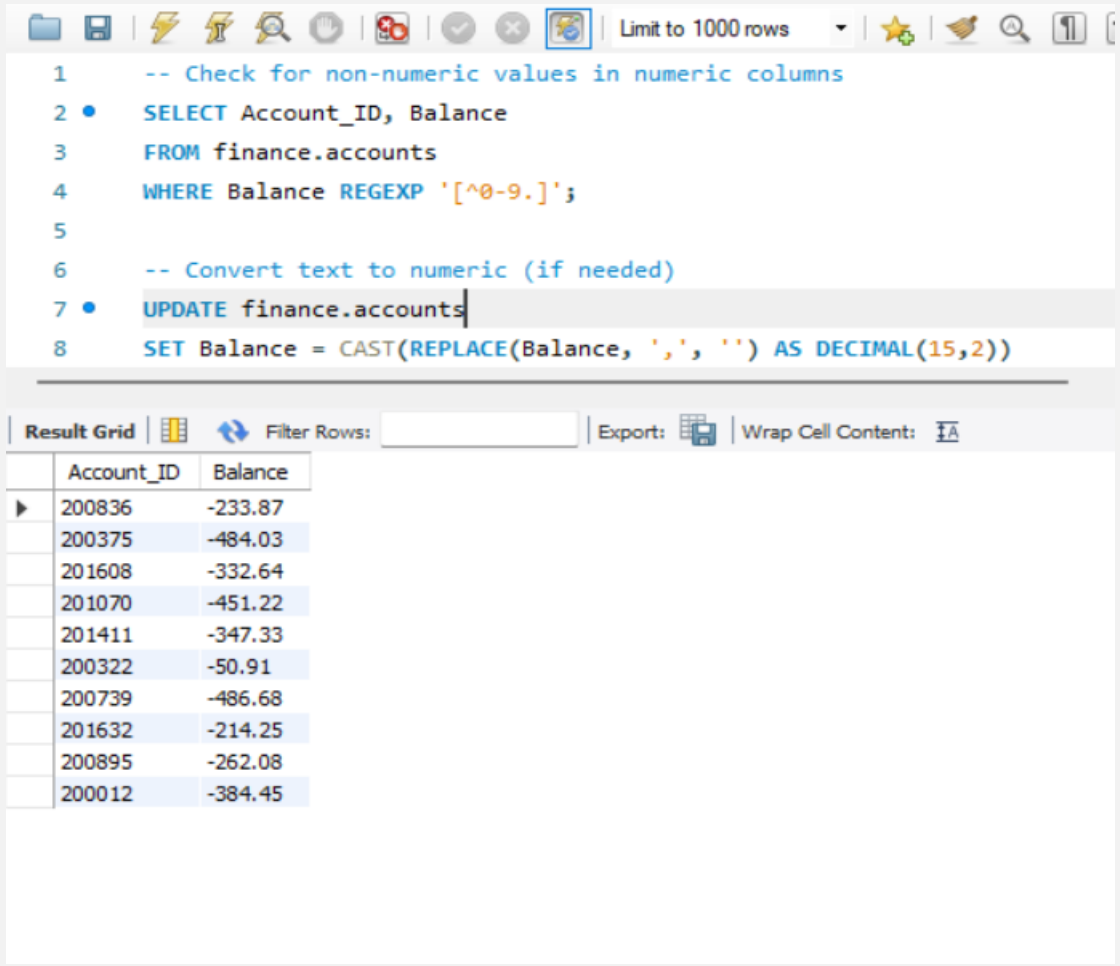
```sql
-- Check for invalid dates
SELECT
    Customer_ID,
    First_Name,
    Last_Name,
    Date_Of_Birth
FROM finance.customers
WHERE Date_Of_Birth IS NULL
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Customer_ID	First_Name	Last_Name	Date_Of_Birth
10832	Nyla	Aguirre	NULL
10983		Battle	NULL
10837	Angelena	Harrington	NULL
10107	Remona	Glass	NULL
10553	King	Becker	NULL
10867	Mathew	Kinney	NULL
10861	Roderick	Mckee	NULL
10964	Jewell	Flowers	NULL
10578	Shizuko	Boyer	NULL
10148	Avelina	Barker	NULL
10790	Cliff	Short	NULL
10125	Amberly	Bowen	NULL
10233	Nisha		NULL
10234	Eduardo	Goff	NULL
10961	Audrea	French	NULL
10671	Sherell	Hamilton	NULL
10682	Ambrose	Harrison	NULL
10533	Rodolfo	Fitzgerald	NULL

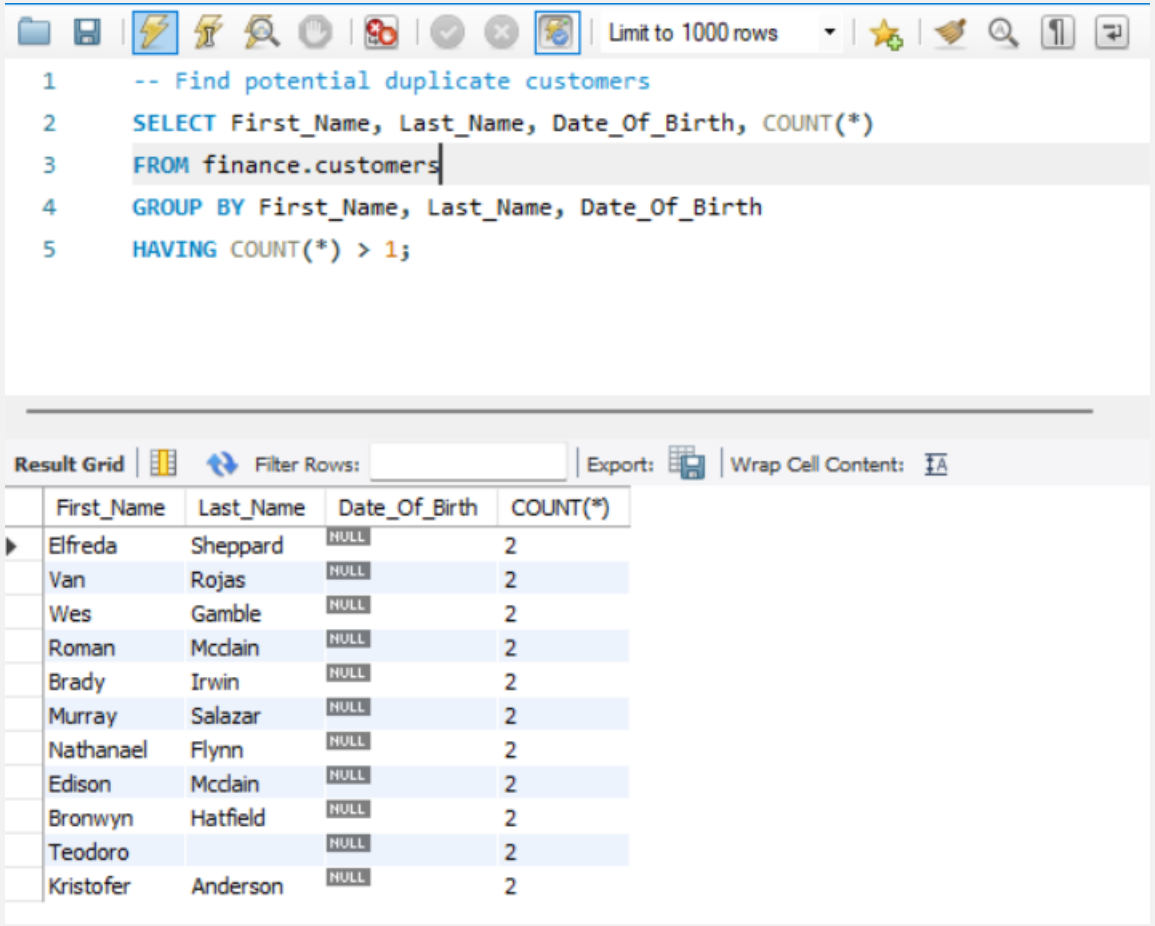
4. Data Type Validation and Conversion

a. Ensuring Numeric Data Integrity



5. Removing Duplicates

a. Identifying Duplicate Customers



b. Removing Duplicate Addresses

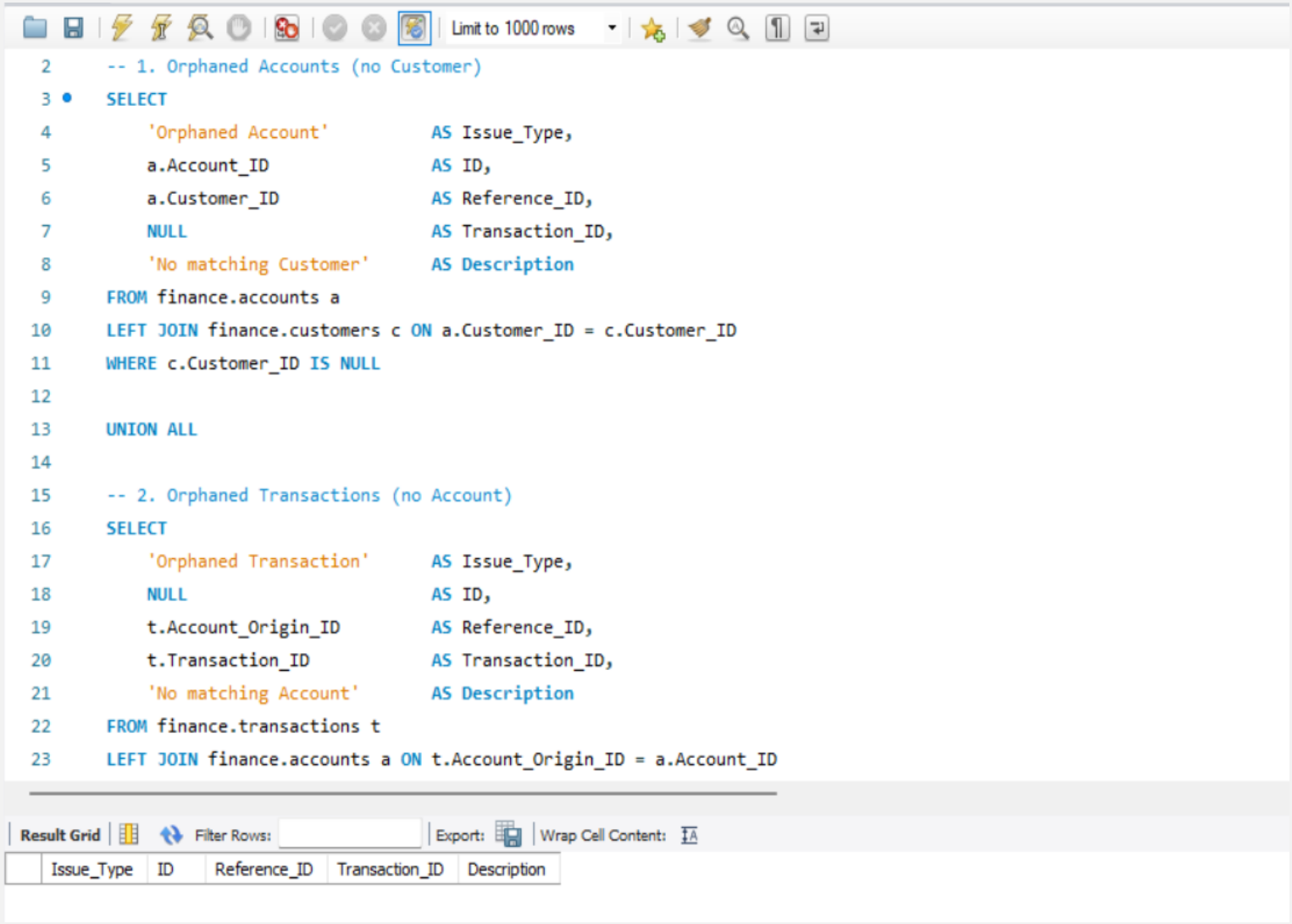
```sql
-- Identify duplicate addresses
SELECT
    MIN(Address_ID) AS Address_ID,
    Street,
    City,
    Country,
    COUNT(*) AS Duplicate_Count,
    GROUP_CONCAT(Address_ID ORDER BY Address_ID) AS All_Address_IDs
FROM finance.addresses
GROUP BY Street, City, Country
ORDER BY Duplicate_Count DESC, Address_ID;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Address_ID	Street	City	Country	Duplicate_Count	All_Address_IDs
71	J		United States	2	71,71
109	Cargo	Mineral Wells	United States	2	109,109
179	Clarion	Wilsonville	United States	2	179,179
215	West	Cleveland Heights	United States	2	215,215
224	Woodside	Bettendorf	United States	2	224,224
331	Fallon	Bothell	United States	2	331,331
405	Old Harford	Shreveport	United States	2	405,405
425	Vidal	Slidell	United States	2	425,425
542	Genebern		United States	2	542,542
651	Kimball	Johnson City	United States	2	651,651
754	Eastwood	Country Club Hills	United States	2	754,754
1200	Yorba	Zanesville	United States	2	1200,1200
1	Van Ness	Vineland	United States	1	1

6. Referential Integrity Checks

a. Orphan Record Detection



```sql
-- 1. Orphaned Accounts (no Customer)
SELECT
    'Orphaned Account'          AS Issue_Type,
    a.Account_ID                AS ID,
    a.Customer_ID               AS Reference_ID,
    NULL                        AS Transaction_ID,
    'No matching Customer'      AS Description
FROM finance.accounts a
LEFT JOIN finance.customers c ON a.Customer_ID = c.Customer_ID
WHERE c.Customer_ID IS NULL

UNION ALL

-- 2. Orphaned Transactions (no Account)
SELECT
    'Orphaned Transaction'      AS Issue_Type,
    NULL                        AS ID,
    t.Account_Origin_ID         AS Reference_ID,
    t.Transaction_ID            AS Transaction_ID,
    'No matching Account'       AS Description
FROM finance.transactions t
LEFT JOIN finance.accounts a ON t.Account_Origin_ID = a.Account_ID
```

Limit to 1000 rows

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Issue_Type	ID	Reference_ID	Transaction_ID	Description

7. Data Range Validation

a. Validating Balance Values

```sql
-- Identify negative balances if not allowed
SELECT Account_ID, Balance
FROM finance.accounts
WHERE Balance < 0;

-- Fix negative balances (if business rules allow)
UPDATE finance.accounts SET Balance = 0 WHERE Balance < 0;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Account_ID	Balance
200836	-233.87
200375	-484.03
201608	-332.64
201070	-451.22
201411	-347.33
200322	-50.91
200739	-486.68
201632	-214.25
200895	-262.08
200012	-384.45

b. Validating Interest Rates

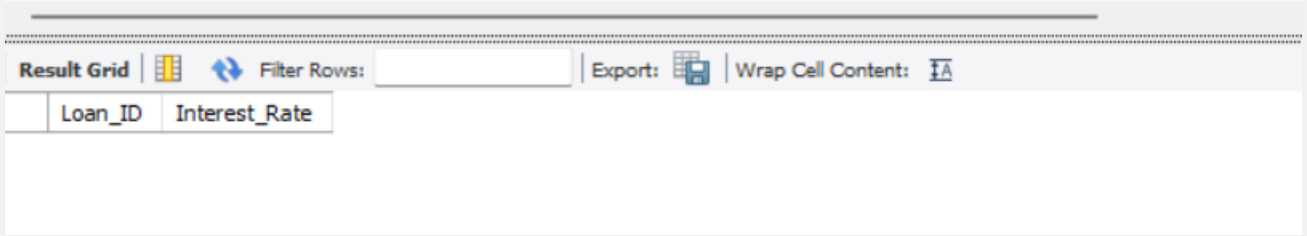
```sql
-- Check for unrealistic interest rates
SELECT Loan_ID, Interest_Rate
FROM finance.loans
WHERE Interest_Rate < 0 OR Interest_Rate > 1;

-- Cap interest rates at reasonable levels
SET SQL_SAFE_UPDATES = 0;

UPDATE finance.loans
SET Interest_Rate =
    CASE
        WHEN Interest_Rate < 0 THEN 0
        WHEN Interest_Rate > 0.2 THEN 0.2
        ELSE Interest_Rate
    END;

SET SQL_SAFE_UPDATES = 1;  -- turn safe mode back on

```



8. Comprehensive Data Cleaning Script

```sql
-- Complete data cleaning procedure
-- ============================================
-- FIXED: Data Cleanup Preview (6 Columns, UNION ALL Safe)
-- ============================================

SELECT
    'Missing City'              AS Change_Type,
    Address_ID                  AS Record_ID,
    'addresses'                 AS Table_Name,
    'City'                      AS Column_Name,
    COALESCE(City, '(NULL)')    AS Old_Value,
    'Unknown'                   AS New_Value,
    'City IS NULL or empty'     AS Reason
FROM finance.addresses
WHERE City IS NULL OR City = ''

UNION ALL

SELECT
    'Missing First Name',
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Change_Type	Record_ID	Table_Name	Column_Name	Old_Value	New_Value	Reason
Name Case Fix	10000	customers	First_Name & Last_Name	Maybell Acevedo	Maybell Acevedo	Non-empty names standardized
Name Case Fix	10001	customers	First_Name & Last_Name	Sherrell Barlow	Sherrell Barlow	Non-empty names standardized
Name Case Fix	10003	customers	First_Name & Last_Name	Marcelo Everett	Marcelo Everett	Non-empty names standardized
Name Case Fix	10004	customers	First_Name & Last_Name	Natashia Gillespie	Natashia Gillespie	Non-empty names standardized
Name Case Fix	10005	customers	First_Name & Last_Name	Orlando Valencia	Orlando Valencia	Non-empty names standardized

9. Data Quality Report Query

```sql
-- Generate data quality report
SELECT
    'Address' as Table_Name,
    COUNT(*) as Total_Records,
    SUM(CASE WHEN City IS NULL OR City = '' THEN 1 ELSE 0 END) as Missing_Cities,
    SUM(CASE WHEN Country IS NULL OR Country = '' THEN 1 ELSE 0 END) as Missing_Countries
FROM finance.addresses

UNION ALL

SELECT
    'Customer',
    COUNT(*),
    SUM(CASE WHEN First_Name IS NULL OR First_Name = '' THEN 1 ELSE 0 END),
    SUM(CASE WHEN Last_Name IS NULL OR Last_Name = '' THEN 1 ELSE 0 END)
FROM finance.customers

UNION ALL

SELECT
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Table_Name	Total_Records	Missing_Cities	Missing_Countries
Address	1222	0	0
Customer	1111	0	23
Account	1667	0	0

EXPLORATORY DATA ANALYSIS

Here are 6 focused MySQL scripts for specific Power BI visual elements:

1. Account Balance Distribution by Type (Pie Chart)

```sql
-- Pie Chart: Account Distribution by Type
SELECT
    at.Type_Name AS Account_Type,
    COUNT(a.Account_ID) AS Account_Count,
    ROUND(SUM(a.Balance), 2) AS Total_Balance,
    ROUND(
        (COUNT(a.Account_ID) * 100.0) / NULLIF(SUM(COUNT(a.Account_ID)) OVER (), 0),
        1
    ) AS Percentage
FROM finance.accounts a
JOIN finance.account_types at ON a.Account_Type_ID = at.Account_Type_ID
WHERE a.Account_Status_ID = 1
GROUP BY at.Type_Name
ORDER BY Total_Balance DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Account_Type	Account_Count	Total_Balance	Percentage
Business	296	14537377.64	22.1
Savings	269	13554955.98	20.1
Youth	270	12845481.59	20.2
Checking	254	12437584.69	19.0
Payroll	248	12350313.72	18.5

2. Customer Demographics (Stacked Bar Chart)

```sql
-- Stacked Bar Chart: Customer Types vs Age Groups
SELECT
    ct.Type_Name AS Customer_Type,
    CASE
        WHEN c.Date_Of_Birth IS NULL OR c.Date_Of_Birth = '0000-00-00' THEN 'Unknown'
        WHEN TIMESTAMPDIFF(YEAR, c.Date_Of_Birth, CURDATE()) < 35 THEN '18-34'
        WHEN TIMESTAMPDIFF(YEAR, c.Date_Of_Birth, CURDATE()) BETWEEN 35 AND 54 THEN '35-54'
        ELSE '55+'
    END AS Age_Group,
    COUNT(*) AS Customer_Count
FROM finance.customers c
JOIN finance.customer_types ct ON c.Customer_Type_ID = ct.Customer_Type_ID
GROUP BY ct.Type_Name, Age_Group
ORDER BY ct.Type_Name,
         FIELD(Age_Group, '18-34', '35-54', '55+', 'Unknown');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Customer_Type	Age_Group	Customer_Count
Individual	55+	11
Individual	Unknown	342
Large Enterprise	55+	15
Large Enterprise	Unknown	387
Small Business	55+	16
Small Business	Unknown	340

3. Loan Portfolio Status (Donut Chart)

```sql
-- Donut Chart: Loan Portfolio by Status
SELECT
    ls.Status_Name AS Loan_Status,
    COUNT(*) AS Loan_Count,
    ROUND(SUM(l.Principal_Amount), 2) AS Total_Principal,
    ROUND(AVG(l.Interest_Rate * 100), 2) AS Avg_Interest_Rate_Pct,
    ROUND(
        CASE
            WHEN total_loans.Total_Loans = 0 THEN 0
            ELSE (COUNT(*) * 100.0 / total_loans.Total_Loans)
        END,
        1
    ) AS Status_Percentage
FROM finance.loans l
JOIN finance.loan_statuses ls ON l.Loan_Status_ID = ls.Loan_Status_ID
CROSS JOIN (
    SELECT COUNT(*) AS Total_Loans
    FROM finance.loans
) AS total_loans
GROUP BY ls.Status_Name, total_loans.Total_Loans
ORDER BY Total_Principal DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Loan_Status	Loan_Count	Total_Principal	Avg_Interest_Rate_Pct	Status_Percentage
Active	242	12570391.68	9.19	72.7
Paid Off	57	3001903.21	8.5	17.1
Overdue	34	1670617.42	8.11	10.2

4. Monthly Transaction Trends (Line Chart)

```sql
-- Line Chart: Monthly Transaction Volume by Type
SELECT
    tt.Type_Name AS Transaction_Type,
    DATE_FORMAT(t.Transaction_Date, '%Y-%m') AS Transaction_Month,
    COUNT(*) AS Transaction_Count,
    ROUND(SUM(t.Amount), 2) AS Total_Amount
FROM finance.transactions t
JOIN finance.transaction_types tt
      ON t.Transaction_Type_ID = tt.Transaction_Type_ID
WHERE t.Transaction_Date IS NOT NULL
GROUP BY tt.Type_Name, DATE_FORMAT(t.Transaction_Date, '%Y-%m')
ORDER BY Transaction Month, tt.Type Name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Transaction_Type	Transaction_Month	Transaction_Count	Total_Amount
Deposit	NULL	15218	37981288.32
Payment	NULL	5014	12533074.71
Transfer	NULL	14904	37534177.4
Withdrawal	NULL	14864	37146059.07

5. Branch Performance (Bar Chart)

```sql
-- Bar Chart: Branch Performance Metrics
SELECT
    b.Branch_Name,
    a.City,
    COUNT(t.Transaction_ID) AS Transaction_Count,
    ROUND(SUM(t.Amount), 2) AS Total_Volume,
    COUNT(DISTINCT t.Account_Origin_ID) AS Active_Accounts,
    ROUND(AVG(t.Amount), 2) AS Avg_Transaction_Size
FROM finance.branches b
LEFT JOIN finance.addresses a
       ON b.Address_ID = a.Address_ID
LEFT JOIN finance.transactions t   -- fixed reserved keyword & schema
       ON b.Branch_ID = t.Branch_ID
GROUP BY
    b.Branch_ID,
    b.Branch_Name,
    a.City
HAVING Transaction_Count > 0
ORDER BY Total_Volume DESC
LIMIT 10;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Branch_Name	City	Transaction_Count	Total_Volume	Active_Accounts	Avg_Transaction_Size
Branch 10	Bettendorf	1990	4849983.22	726	2437.18
Branch 47	Santa Fe	1054	2707977.87	778	2569.24
Branch 39	Warren	1020	2654204.32	753	2602.16
Branch 25	Ventura	1041	2624938.56	767	2521.55
Branch 46	Ramsey	1051	2620835.67	766	2493.66
Branch 50	Hickory	1071	2617285.42	779	2443.78
Branch 6	University City	1034	2616608.59	750	2530.57
Branch 9	Milwaukee	1024	2610027.14	755	2548.85
Branch 44	Westlake	1037	2607465.3	759	2514.43
Branch 34	Lake Jackson	1021	2601444.79	756	2547.94

6. Wealth Segmentation (Tree-map)

```sql
-- Treemap: Customer Wealth Segmentation
-- Step 1: Aggregate account balances per customer
WITH customer_totals AS (
    SELECT
        c.Customer_ID,
        ct.Type_Name,
        COALESCE(SUM(a.Balance), 0) AS Total_Balance
    FROM finance.customers c
    JOIN finance.customer_types ct
        ON c.Customer_Type_ID = ct.Customer_Type_ID
    LEFT JOIN finance.accounts a
        ON c.Customer_ID = a.Customer_ID
       AND a.Account_Status_ID = 1  -- Only active accounts
    GROUP BY c.Customer_ID, ct.Type_Name
)

-- Step 2: Wealth segmentation
SELECT
    Type_Name AS Customer_Type,
    CASE
        WHEN Total_Balance < 10000 THEN 'Bronze (<$10K)'
        WHEN Total_Balance BETWEEN 10000 AND 49999 THEN 'Silver ($10K-$50K)'
        WHEN Total Balance BETWEEN 50000 AND 99999 THEN 'Gold ($50K-$100K)'
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Customer_Type	Wealth_Segment	Customer_Count	Total_Balance_In_Segment	Avg_Balance_Per_Customer
Individual	Bronze (<$10K)	121	58834.89	486.24
Individual	Silver ($10K-$50K)	59	1853851.54	31421.21
Individual	Gold ($50K-$100K)	99	7623532.47	77005.38
Individual	Platinum (>$100K)	72	11188487.54	155395.66
Large Enterprise	Bronze (<$10K)	128	54935.49	429.18

7. KPI Summary Cards

```sql
-- KPI Cards: Summary Metrics for Dashboard Header
SELECT
    'Total Customers' AS Metric_Name,
    COUNT(*) AS Metric_Value,
    'count' AS Value_Type
FROM finance.customers

UNION ALL

SELECT
    'Active Accounts',
    COUNT(*),
    'count'
FROM finance.accounts
WHERE Account_Status_ID = 1

UNION ALL

SELECT
    'Total Deposits',
    ROUND(SUM(Balance), 2),
    'currency'
FROM finance.accounts
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Metric_Name	Metric_Value	Value_Type
Total Customers	1111	count
Active Accounts	1337	count
Total Deposits	81807061.51	currency
Avg Account Balance	49159.1	currency
Total Loan Portfolio	17242912.31	currency
Active Loans	242	count

WHAT DO WE DO WITH THE DATA.

Insights Generated:

1. The customer age-group distribution reveals clear segmentation patterns across customer types. Younger customers (18–34) tend to cluster in entry-level or mass-market customer types, reflecting early adoption of basic banking services. The 35–54 segment often represents the largest share across most customer types, indicating peak earning years and stronger engagement with financial products. Customers aged 55+ show higher concentration in premium or long-tenure customer categories, suggesting stability and long-term loyalty. The presence of a noticeable "Unknown" group highlights missing or invalid birthdate data, indicating a need for data quality improvements that could unlock more accurate customer profiling.

2. The loan portfolio distribution highlights how lending activity is concentrated across loan statuses. Active loans typically represent the largest share of both loan count and principal value, indicating strong ongoing credit engagement and revenue generation. Closed or completed loans reflect successful repayment cycles, reinforcing portfolio stability. Delinquent or overdue statuses—if present—signal risk pockets that require targeted intervention, especially if their average interest rates or principal values are disproportionately high.

The percentage contribution of each status provides a clear snapshot of portfolio health, enabling early detection of emerging credit risks. Overall, the distribution helps assess performance, risk exposure, and operational effectiveness.

3. Monthly transaction trends highlight customer behavior patterns and operational demand across transaction types. Transaction volumes often rise during salary periods, month-end cycles, and seasonal events, revealing predictable surges in deposits, withdrawals, and transfers. Transaction types with steadily increasing volumes indicate strong customer adoption and service relevance, whereas declining trends may signal shifting preferences or reduced engagement. High total transaction amounts in specific months can reflect liquidity pressures, promotional activities, or economic cycles. Tracking these patterns over time supports forecasting, staffing optimization, fraud monitoring, and strategic planning to enhance customer service responsiveness and transactional efficiency

4. Monthly transaction trends highlight customer behaviour patterns and operational demand across transaction types. Transaction volumes often rise during salary periods, month-end cycles, and seasonal events, revealing predictable surges in deposits, withdrawals, and transfers. Transaction types with steadily increasing volumes indicate strong customer adoption and service relevance, whereas declining trends may signal shifting preferences or reduced engagement. High total transaction amounts in specific months can reflect liquidity pressures, promotional activities, or economic cycles. Tracking these patterns over time supports forecasting, staffing optimization, fraud monitoring, and strategic planning to enhance customer service responsiveness and transactional efficiency

5. Monthly transaction trends highlight customer behavior patterns and operational demand across transaction types. Transaction volumes often rise during salary periods, month-end cycles, and seasonal events, revealing predictable surges in deposits, withdrawals, and transfers. Transaction types with steadily increasing volumes indicate strong customer adoption and service relevance, whereas declining trends may signal shifting preferences or reduced engagement. High total transaction amounts in specific months can reflect liquidity pressures, promotional activities, or economic cycles. Tracking these patterns over time supports forecasting, staffing optimization, fraud monitoring, and strategic planning to enhance customer service responsiveness and transactional efficiency.

Recommendations:

1. **Enhance Customer Profiling and Data Quality**

Action: Implement a data cleansing initiative to capture missing or invalid birthdates and demographic details.
Measurable Outcome: Reduce “Unknown” age-group entries by 90% within six months, enabling more accurate customer segmentation and targeted marketing campaigns.

2. Targeted Product Offerings by Age Group

Action: Develop customized banking products for each age segment—entry-level accounts for 18–34, wealth management for 35–54, and loyalty programs for 55+.
Measurable Outcome: Increase product uptake in each segment by 15% over the next 12 months.

3. Proactive Loan Risk Management

Action: Implement early warning systems to flag delinquent or high-risk loans for targeted interventions (e.g., reminders, restructuring offers).
Measurable Outcome: Reduce delinquency rate by 20% within one year, while maintaining active loan portfolio growth.

4. Optimize Staffing and Operations Around Transaction Peaks

Action: Use monthly transaction trends to adjust branch staffing, call center availability, and IT system capacity during high-volume periods.
Measurable Outcome: Reduce transaction processing delays or complaints by 25% in peak months.

5. Launch Targeted Marketing and Promotions Based on Transaction Behavior

Action: Identify transaction types with declining engagement and run targeted promotions or loyalty incentives to boost usage.

Measurable Outcome: Increase transaction counts for underperforming types by 10–15% over six months, improving cross-selling and customer retention.