DATA CLEANING AND ANALYZING IN SQL

SQL Data Cleaning & Transformation

This section outlines the steps taken to clean and transform the raw data files (transactions, customers, agents, calendar, sales_targets) within the SQL database.

1. Column & Table Standardization

The first step was to standardize table and column names for consistency and readability, converting all names to lowercase and fixing spelling errors.

- **Table Renaming:**

```
ALTER TABLE transaction RENAME TO transactions;

ALTER TABLE calendar RENAME TO calender;
```

- **Column Renaming:** Renamed columns to ensure a consistent naming convention (snake_case) and correct spelling.

```
-- Example for the 'customers' table

ALTER TABLE customers RENAME COLUMN customerid TO customer_id;

ALTER TABLE customers RENAME COLUMN signupdate TO signup_date;
```

- **Case Conversion:** A query was used to generate ALTER TABLE statements to convert all column names to lowercase.

```
-- Example query to generate statements

SELECT 'ALTER TABLE ' || table_name || ' RENAME COLUMN "' || column_name || '" TO ' ||
LOWER(column_name) || ';'

FROM information_schema.columns

WHERE table_name = 'transactions';
```

---

2. Data Type & Value Conversion

Ensured that data types were correct and values were standardized.

- **Data Type Alteration:** Changed the isweekend column to TEXT for easier value manipulation.

```
ALTER TABLE calendar ALTER COLUMN isweekend TYPE TEXT USING isweekend :: TEXT;
```

- **Value Standardization:** Converted boolean values to standardized Yes/No strings.

```
UPDATE calendar SET isweekend = 'No' WHERE isweekend = 'false';

UPDATE calendar SET isweekend = 'Yes' WHERE isweekend = 'true';
```

- **Value Truncation:** Standardized month and weekday names to a 3-letter format.

```
UPDATE calendar SET dayofweek = LEFT(dayofweek, 3);

UPDATE calendar SET month_name = LEFT(month_name, 3);
```

---

3. Handling Duplicates & Missing Data

Identified and resolved data quality issues like duplicate rows and null values.

- **Checking for Nulls:** Queries were run to count null values in key columns.

```
SELECT COUNT(*) FILTER(WHERE transaction_id ISNULL) AS transaction_id_nulls FROM
transactions;
```

- **Removing Duplicates:** Used a window function to identify and remove duplicate rows based on a unique identifier.

```
DELETE FROM Transactions WHERE transaction_id IN (

   SELECT transaction_id FROM (

      SELECT transaction_id, ROW_NUMBER() OVER(PARTITION BY transaction_id ORDER BY
transaction_id) AS rn

      FROM Transactions

   ) t WHERE rn > 1

);
```

- **Imputing Missing Values:** Used a MODE function with a WITH clause to impute missing state values based on the most frequent state per country.

```
WITH t AS ( ... )

UPDATE customers SET state = mrs FROM t WHERE state IS NULL AND customers.country =
t.country;
```

---

4. Outlier Detection

Examined the range of values in key columns to identify potential outliers that could skew analysis.

- **Range Checks:** Queried the minimum and maximum values for amounts, fees, and dates.

```
SELECT MIN(fee_amount), MAX(fee_amount) FROM transactions;
```

SQL Exploratory Data Analysis (EDA)

This section details the key queries used to extract insights from the cleaned data.

1. Overall Business Performance

Calculated high-level metrics to understand the overall health of the business.

- **Total Transactions & Revenue:**

```
SELECT COUNT(DISTINCT transaction_id) AS total_transactions,
    ROUND(SUM(transaction_amount) :: numeric, 2) AS total_transaction_amount,
    ROUND(SUM(fee_amount) ::numeric, 2) AS revenue
FROM transactions WHERE status = 'Completed';
```

---

2. Performance by Segment

Segmented the data to understand performance across different dimensions.

- **Monthly Trends:**

```
SELECT month_name, COUNT(DISTINCT transaction_id) AS monthly_transactions,
    ROUND(SUM(fee_amount) :: numeric, 2) AS monthly_revenue
FROM transactions WHERE status = 'Completed' GROUP BY month_name;
```

- **Top 10 Customers:**

```
SELECT sender_customerid, customers.name, COUNT(transaction_id) AS total_transactions,
    ROUND(SUM(transaction_amount) :: numeric, 2) AS total_trac_amount
FROM transactions JOIN customers ON transactions.sender_customerid =
customers.customer_id
GROUP BY sender_customerid, name ORDER BY total_trac_amount DESC LIMIT 10;
```

- **Agent Performance:**

```
SELECT a.agent_name, a.agent_id, COUNT(DISTINCT t.transaction_id) tran_count,
    SUM(t.fee_amount) total_revenue
FROM transactions t JOIN agents a ON t.agent_id = a.agent_id
WHERE t.status = 'Completed' GROUP BY a.agent_name, a.agent_id ORDER BY total_revenue DESC;
```

---

3. Operational Insights

Analyzed operational metrics to identify trends in business processes.

- **Payment Method Analysis:**

```
SELECT payment_method, COUNT(DISTINCT transaction_id) total_transactions,
    SUM(fee_amount) revenue
FROM transactions WHERE status = 'Completed' GROUP BY payment_method ORDER BY revenue DESC;
```

- **Weekend vs. Weekday Activity:**

```
SELECT CASE WHEN isweekend = 'Yes' THEN 'weekend' ELSE 'weekday' END AS week,
    COUNT(DISTINCT transaction_id) tran_count
FROM transactions WHERE status = 'Completed' GROUP BY week;
```

---

4. Target vs. Actual Performance

Joined transactions with sales_targets and agents to compare actual performance against set goals.

- **Agent-Level Performance Comparison:**

```
SELECT st.Month, st.Agent_ID, a.Agent_Name,
    COUNT(t.Transaction_ID) AS actual_transactions,
    SUM(t.Fee_Amount) AS actual_revenue,
    st.Target_Transactions, st.Target_Revenue
FROM Sales_Targets st
JOIN Agents a ON st.Agent_ID = a.Agent_ID
```

```sql
LEFT JOIN Transactions t ON st.Agent_ID = t.Agent_ID AND EXTRACT(MONTH FROM t.Transaction_Date) = st.Month

GROUP BY st.Month, st.Agent_ID, a.Agent_Name, st.Target_Transactions, st.Target_Revenue;
```