Data QA - Code Challenge

KAREEM ALNABULSI

user

Contents

[QA Report 2](#_Toc187708201)

[Challenge One 2](#_Toc187708202)

[Test Scope 2](#_Toc187708203)

[Implementation 2](#_Toc187708204)

[Challenge Two 4](#_Toc187708205)

[Test Scope 4](#_Toc187708206)

[Implementation 4](#_Toc187708207)

# QA Report

This document contains a detailed report of the topics that were to be tested, alongside a brief explanation of the testing process and the solutions.

## Challenge One

The customer data provided consists of the following columns:

* ID
* First Name
* Last Name
* Address
* City
* State
* Zip Code
* Phone Number
* Email
* Birthdate

### Test Scope

The scope of work was to test and validate this data, ensuring data integrity. The following list showcases the validation criteria followed:

1. **Duplicated Data:** Check if any of the rows are duplicated by ensuring that the first name, last name, email, and phone number are unique and only recur once.
2. **Missing Data:** Check if any of the rows include missing data (NULL).
3. **Invalid email formats:** Check if the emails are correctly formatted according to the following rule: **%@%.%**
4. **Invalid phone number formats:** Check if the phone number formats match the following rule: **‘\_\_\_-\_\_\_-\_\_\_\_'**
5. **Inconsistent Data:** Check if there are any duplicate users that have the first name and last name, but different information such as email and address.

### Implementation

The following SQL queries were used to validate data integrity:

1. Duplicated Data:

*SELECT first\_name, last\_name, email, phone\_number, COUNT(\*) AS duplicate\_count*

*FROM customer\_table*

*GROUP BY first\_name, last\_name, email, phone\_number*

*HAVING COUNT(\*) > 1;*

1. Missing Data:

*SELECT \**

*FROM customer\_table*

*WHERE first\_name IS NULL*

*OR last\_name IS NULL*

*OR address IS NULL*

*OR city IS NULL*

*OR state IS NULL*

*OR zip\_code IS NULL*

*OR phone\_number IS NULL*

*OR email IS NULL*

*OR birthdate IS NULL;*

1. Invalid Emails Formats:

*SELECT id, email*

*FROM customer\_table*

*WHERE email NOT LIKE '%@%.%';*

1. Invalid Phone Number Formats:

*SELECT id, phone\_number*

*FROM customer\_table*

*WHERE phone\_number NOT LIKE '\_\_\_-\_\_\_-\_\_\_\_';*

1. Inconsistent\_Data:

*SELECT first\_name, last\_name, COUNT(DISTINCT email) AS unique\_emails, COUNT(DISTINCT address) AS unique\_addresses*

*FROM customer\_table*

*GROUP BY first\_name, last\_name*

*HAVING COUNT(DISTINCT email) > 1 OR COUNT(DISTINCT address) > 1;*

#### Recommendation / Test Results

The results in the excel sheet “Challenge One – Results.xlsx” show that this table should not be shared with the stakeholders in its current state, as the test revealed multiple discrepancies which have affected data integrity and quality and it cannot be used in its current state as there are multiple missing values, as well as incorrectly formatted values.

Given that the data quality is compromised, the stakeholders will not benefit from viewing this table as it is lacking information or includes incorrect information.

## Challenge Two

Tables L1 and L2 have been provided inside the excel sheet “QA code challange Question 2 - Services .xlsx”. Python was used in order to insert both these tables into a local postgreSQL database. This was done as follows:

**# Read both sheets into separate DataFrames**

**df\_sheet1 = pd.read\_excel(excel\_file, sheet\_name='l1\_services')**

**df\_sheet2 = pd.read\_excel(excel\_file, sheet\_name='l2\_services')**

**# Create the tables in the database:**

**df\_sheet1.to\_sql('l1', connection\_string, if\_exists='replace', index=False)**

**df\_sheet2.to\_sql('l2', connection\_string, if\_exists='replace', index=False)**

First, we read both sheets using pandas and place them into data frames. After the data frames have been created, we use the **.to\_sql** function to create the tables, passing along the database credentials, and extra parameters.

### Test Scope

The scope of work for this challenge was to validate the correct transformation of L1 into L2 based on business requirements. The transformation follows the following rules:

1. **Filtering**: Data from L1 is filtered to ensure that only rows with product\_type equal to order\_type are included for all columns except for *service\_fee\_code*.
2. **Transformation**: For each row in L1, the data is transformed and mapped to the corresponding business attributes:
   * If *product\_type == 'rule'*, service\_fee\_code is set to the product\_name.
   * **Calculated Fields**: Fields like iov\_usd and gbv\_usd are calculated using the formula *value \* conversion\_rate\_usd*, and the decimal values are trimmed to two decimal places using the trim\_decimal() function.

To complete this test, L1 will need to be transformed into L2 based on the business requirments, after that the data in L1 will be used to validate the accuracy of the data in L2.

### Implementation

1. **Fetch data from database:**  
   *# SQL queries to fetch the data from L1 and L2 tables*

*L1\_query = "SELECT \* FROM l1;"*

*L2\_query = "SELECT \* FROM l2;"  
# Fetch data from L1*

*L1 = connection.execute(text(L1\_query))*

*# Fetch all the results and convert them into a DataFrame*

*allResults = L1.fetchall()*

*L1 = pd.DataFrame(allResults, columns=L1.keys())*

*# Fetch data from L2*

*L2 = connection.execute(text(L2\_query))*

*# Fetch all the results and convert them into a DataFrame*

*allResults = L2.fetchall()*

*L2 = pd.DataFrame(allResults, columns=L2.keys())*

1. **Filter un-used columns:**

*filtered\_L1 = L1[L1['product\_type'] == L1['order\_type']]*

1. **Data Transformation:** Transform the filtered L1 data into a structured format (transformed\_L1\_df) that aligns with the business requirements for L2.

*transformed\_L1 = []*

*for \_, row\_L1 in filtered\_L1.iterrows():*

*transformed\_row = {*

*'order\_type': row\_L1['order\_type'],*

*'dim\_group\_id': row\_L1['dim\_group\_id'],*

*'order\_no': row\_L1['order\_no'],*

*'dim\_bookingdate\_id': row\_L1['dim\_bookingdate\_id'],*

*'dim\_store\_id': row\_L1['dim\_store\_id'],*

*'service\_fee\_code': row\_L1['product\_name'] if row\_L1['product\_type'] == 'rule' else None,*

*'dim\_customer\_id': row\_L1['dim\_customer\_id'],*

*'dim\_language': row\_L1['dim\_language'],*

*'dim\_totals\_currency': row\_L1['dim\_totals\_currency'],*

*'dim\_status\_id': row\_L1['dim\_status\_id'],*

*'phone': row\_L1['phone'],*

*'payment\_amount': row\_L1['payment\_amount'],*

*'discount\_amount': row\_L1['discount\_amount'],*

*'service\_fee\_amount': row\_L1['service\_fee\_amount'],*

*'base\_amount': row\_L1['base\_amount'],*

*'inputvat': row\_L1['inputvat'],*

*'outputvat': row\_L1['outputvat'],*

*'product\_vat': row\_L1['product\_vat'],*

*'selling\_price': row\_L1['selling\_price'],*

*'selling\_price\_vat': row\_L1['selling\_price\_vat'],*

*'ibv': row\_L1['ibv'],*

*'iov\_usd': trim\_decimal(row\_L1['ibv'] \* row\_L1['conversion\_rate\_usd']),*

*'gbv': row\_L1['gbv'],*

*'gbv\_usd': trim\_decimal(row\_L1['gbv'] \* row\_L1['conversion\_rate\_usd']),}*

*transformed\_L1.append(transformed\_row)*

*transformed\_L1\_df = pd.DataFrame(transformed\_L1)*

1. **Validate L2:** Compare transformed\_L1\_df with L2 to ensure data accuracy and alignment with the business requirements.

* **Row Matching:** Each row in L2 is matched against the corresponding row in transformed\_L1\_df based on the order\_no.
* **Column Validation:** For each matching row, values in all columns are compared using the are\_values\_different() function to identify mismatches.
* **Error Logging:** Mismatched rows or columns are logged for further review.

errors = []

for \_, row\_L2 in L2.iterrows():

# Find matching row in transformed\_L1\_df

matching\_rows = transformed\_L1\_df[transformed\_L1\_df['order\_no'] == row\_L2['order\_no']]

if matching\_rows.empty:

errors.append({'order\_no': row\_L2['order\_no'], 'error': 'No matching row in L1'})

else:

# Compare each column

for col in matching\_rows.columns:

value\_L2 = str(row\_L2[col])

value\_L1 = str(matching\_rows.iloc[0][col])

if are\_values\_different(value\_L1, value\_L2):

errors.append({

'order\_no': row\_L2['order\_no'],

'column': col,

'L1\_value': value\_L1,

'L2\_value': value\_L2

})

#### Recommendation / Test Results

Running this script indicated a total of 412 mismatches as show in the excel sheet “Challenge Two – Results.xlsx” between L2 and the required business logic. Thus, this does not meet the requirements. The errors are mainly as follows:

1. Incorrect values for service\_fee\_code in L2, indicating the conditions set by business have not been set on the transformed table L2.
2. Incorrect values for iov\_usd, indicitating that the wrong equation was used to calculate this value.
3. Incorrect values for gbv\_usd, indicitating that the wrong equation was used to calculate this value.
4. Multiple data mismatches for various columns, indicating the wrong transformation of L2 (ex. Row 274 in results sheet.