**Project: ETL Testing for Credit Card Data Pipeline**

This project simulates an ETL testing process for a credit card data pipeline, showcasing essential skills, tools, and methodologies required for an ETL tester role.

**1. Project Overview**

**Objective:**

To test an ETL pipeline that extracts, transforms, and loads credit card transaction data, ensuring data accuracy, completeness, and compliance.

**Scope:**

* Validate data extraction from source systems.
* Verify transformation logic aligns with business rules.
* Ensure accurate and complete loading into the target data warehouse.
* Perform functional, performance, and reconciliation testing.

**Deliverables:**

* Test cases and test scripts.
* SQL queries for data validation.
* ETL testing reports.

**2. Tools and Technologies**

| **Tool** | **Purpose** |
| --- | --- |
| **SQL (PostgreSQL)** | Write validation queries for data accuracy. |
| **Python** | Automate ETL testing processes. |
| **JIRA** | Track defects and testing progress. |
| **Apache Airflow** | Schedule and monitor ETL workflows. |
| **DBeaver** | Query databases and analyze results. |
| **Excel** | Perform manual validation of sample data. |

**3. ETL Testing Steps**

**3.1 Extract Testing:**

* Validate the data extracted from source systems matches source requirements.
* **Key Validation Queries:**

-- Check row count in source system

SELECT COUNT(\*) FROM source\_transactions;

-- Verify data types and constraints

SELECT COLUMN\_NAME, DATA\_TYPE

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = 'source\_transactions';

**3.2 Transform Testing:**

* Validate transformation logic, such as currency conversion and date formatting.
* **Python Example for Transformation Validation:**

# Validate currency conversion logic

source\_amount = 100 # USD

conversion\_rate = 0.85 # USD to EUR

expected\_result = source\_amount \* conversion\_rate

assert transform\_currency(source\_amount, conversion\_rate) == expected\_result

* **SQL Query to Verify Transformation:**

-- Check if transformed dates match target format

SELECT transaction\_date

FROM target\_transactions

WHERE transaction\_date NOT LIKE 'YYYY-MM-DD';

-- Verify calculated fields

SELECT transaction\_id, reward\_points,

CASE

WHEN amount\_spent >= 100 THEN 10

ELSE 5

END AS expected\_points

FROM target\_transactions;

**3.3 Load Testing:**

* Verify data is loaded correctly into the target system.
* **Key Validation Queries:**

-- Ensure row count matches between source and target

SELECT COUNT(\*) AS source\_count FROM source\_transactions;

SELECT COUNT(\*) AS target\_count FROM target\_transactions;

-- Verify primary key constraints in the target

SELECT transaction\_id, COUNT(\*)

FROM target\_transactions

GROUP BY transaction\_id

HAVING COUNT(\*) > 1;

**4. Types of Testing**

**4.1 Functional Testing:**

* Validate business rules, such as reward points calculation or transaction categorization.

**4.2 Data Validation Testing:**

* Ensure data consistency between source and target systems.

**4.3 Performance Testing:**

* Validate the ETL pipeline processes data within acceptable time limits.

**4.4 Reconciliation Testing:**

* Ensure all records from the source system are loaded into the target system.

**5. Sample Test Cases**

| **Test Case ID** | **Test Scenario** | **Expected Result** |
| --- | --- | --- |
| TC001 | Validate row count matches between source/target | Row counts are equal. |
| TC002 | Check for duplicate records in the target | No duplicate records exist. |
| TC003 | Verify date transformation logic | Dates are in YYYY-MM-DD format. |
| TC004 | Validate reward points calculation | Reward points are calculated as per business rules. |

**6. Project Structure**

**Folder Organization:**

* **/sql\_queries**: SQL scripts for validation.
* **/python\_scripts**: Python scripts for automated testing.
* **/test\_cases**: Documented test cases.
* **/logs**: Logs of ETL runs and test results.

**7. Reporting**

* Generate a daily report with key metrics:
  + Total records processed.
  + Number of failed records.
  + Defects identified and resolved.

**Sample Python Code for Report Automation:**

import pandas as pd

def generate\_report(processed, failed, defects):

data = {

'Metric': ['Records Processed', 'Records Failed', 'Defects'],

'Count': [processed, failed, defects]

}

df = pd.DataFrame(data)

df.to\_csv('etl\_test\_report.csv', index=False)

# Example usage

generate\_report(10000, 25, 3)

**8. Conclusion**

By completing this project, you demonstrate:

* Proficiency in ETL testing methodologies.
* Experience with tools like SQL, Python, and JIRA.
* Ability to write test cases, validate data, and ensure data quality.

Would you like me to provide detailed code files or expand on specific sections?

**Project: ETL Testing for Credit Card Data Pipeline**

This project simulates an ETL testing process for a credit card data pipeline, showcasing essential skills, tools, and methodologies required for an ETL tester role.

**1. Project Overview**

**Objective:**

To test an ETL pipeline that extracts, transforms, and loads credit card transaction data, ensuring data accuracy, completeness, and compliance.

**Scope:**

* Validate data extraction from source systems.
* Verify transformation logic aligns with business rules.
* Ensure accurate and complete loading into the target data warehouse.
* Perform functional, performance, and reconciliation testing.

**Deliverables:**

* Test cases and test scripts.
* SQL queries for data validation.
* ETL testing reports.

**2. Tools and Technologies**

| **Tool** | **Purpose** |
| --- | --- |
| **SQL (PostgreSQL)** | Write validation queries for data accuracy. |
| **Python** | Automate ETL testing processes. |
| **JIRA** | Track defects and testing progress. |
| **Apache Airflow** | Schedule and monitor ETL workflows. |
| **DBeaver** | Query databases and analyze results. |
| **Excel** | Perform manual validation of sample data. |

**3. ETL Testing Steps**

**3.1 Extract Testing:**

* Validate the data extracted from source systems matches source requirements.
* **Key Validation Queries:**

-- Check row count in source system

SELECT COUNT(\*) FROM source\_transactions;

-- Verify data types and constraints

SELECT COLUMN\_NAME, DATA\_TYPE

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = 'source\_transactions';

**3.2 Transform Testing:**

* Validate transformation logic, such as currency conversion and date formatting.
* **Python Example for Transformation Validation:**

# Validate currency conversion logic

def transform\_currency(amount, rate):

return amount \* rate

source\_amount = 100 # USD

conversion\_rate = 0.85 # USD to EUR

expected\_result = source\_amount \* conversion\_rate

assert transform\_currency(source\_amount, conversion\_rate) == expected\_result, "Currency conversion failed"

* **SQL Query to Verify Transformation:**

-- Check if transformed dates match target format

SELECT transaction\_date

FROM target\_transactions

WHERE transaction\_date NOT LIKE 'YYYY-MM-DD';

-- Verify calculated fields

SELECT transaction\_id, reward\_points,

CASE

WHEN amount\_spent >= 100 THEN 10

ELSE 5

END AS expected\_points

FROM target\_transactions;

**3.3 Load Testing:**

* Verify data is loaded correctly into the target system.
* **Key Validation Queries:**

-- Ensure row count matches between source and target

SELECT COUNT(\*) AS source\_count FROM source\_transactions;

SELECT COUNT(\*) AS target\_count FROM target\_transactions;

-- Verify primary key constraints in the target

SELECT transaction\_id, COUNT(\*)

FROM target\_transactions

GROUP BY transaction\_id

HAVING COUNT(\*) > 1;

-- Check data consistency between source and target

SELECT s.transaction\_id, s.amount AS source\_amount, t.amount AS target\_amount

FROM source\_transactions s

LEFT JOIN target\_transactions t

ON s.transaction\_id = t.transaction\_id

WHERE s.amount != t.amount;

**4. Types of Testing**

**4.1 Functional Testing:**

* Validate business rules, such as reward points calculation or transaction categorization.

**4.2 Data Validation Testing:**

* Ensure data consistency between source and target systems.

**4.3 Performance Testing:**

* Validate the ETL pipeline processes data within acceptable time limits.

**4.4 Reconciliation Testing:**

* Ensure all records from the source system are loaded into the target system.

**5. Sample Test Cases**

| **Test Case ID** | **Test Scenario** | **Expected Result** |
| --- | --- | --- |
| TC001 | Validate row count matches between source/target | Row counts are equal. |
| TC002 | Check for duplicate records in the target | No duplicate records exist. |
| TC003 | Verify date transformation logic | Dates are in YYYY-MM-DD format. |
| TC004 | Validate reward points calculation | Reward points are calculated as per business rules. |

**6. Project Structure**

**Folder Organization:**

* **/sql\_queries**: SQL scripts for validation.
* **/python\_scripts**: Python scripts for automated testing.
* **/test\_cases**: Documented test cases.
* **/logs**: Logs of ETL runs and test results.

**7. Reporting**

* Generate a daily report with key metrics:
  + Total records processed.
  + Number of failed records.
  + Defects identified and resolved.

**Sample Python Code for Report Automation:**

import pandas as pd

def generate\_report(processed, failed, defects):

data = {

'Metric': ['Records Processed', 'Records Failed', 'Defects'],

'Count': [processed, failed, defects]

}

df = pd.DataFrame(data)

df.to\_csv('etl\_test\_report.csv', index=False)

# Example usage

generate\_report(10000, 25, 3)

**8. Conclusion**

By completing this project, you demonstrate:

* Proficiency in ETL testing methodologies.
* Experience with tools like SQL, Python, and JIRA.
* Ability to write test cases, validate data, and ensure data quality.

This document provides complete coding, programs, and queries needed for an ETL tester role. Let me know if you'd like further customization or additional examples!