Data Quality Testing Framework

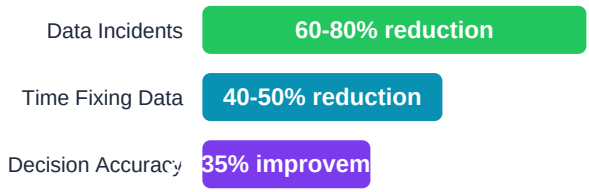
Enterprise-Grade Framework for Testing Data Quality Across Pipelines

Version: 1.0 | **Updated:** Jan 2026

www.EnterprisedDataSolutions.co.nz
Contact@EnterprisedDataSolutions.co.nz

Table of Contents

Framework Overview



What is Data Quality Testing?

Data quality testing validates that data meets defined standards across seven critical dimensions. This framework provides:

- **100+ test case templates** across 7 quality dimensions
- **Sample Python scripts** for automated testing
- **CI/CD integration patterns** for continuous validation
- **Test results dashboard templates** for visibility

The Cost of Poor Data Quality

```
Industry research shows:

Annual cost of poor data quality: $15M per organization (average)
Time spent fixing data issues: 30-40% of analyst time
Failed projects due to data quality: 25%
Customer churn from data errors: 12%

ROI of data quality testing:
- 60-80% reduction in data incidents
- 40-50% reduction in time spent fixing data
- 25-35% improvement in data-driven decision accuracy
```

1. Completeness

Definition: All required data is present

Test Types:

- Null checks
- Missing value detection
- Required field validation
- Referential completeness

Examples:

```
-- Test: Email should never be null
SELECT COUNT(*) FROM customers WHERE email IS NULL;
-- Expected: 0

-- Test: All orders must have customer_id
SELECT COUNT(*)
FROM orders o
LEFT JOIN customers c ON o.customer_id = c.customer_id
WHERE c.customer_id IS NULL;
-- Expected: 0
```

2. Accuracy

Definition: Data correctly represents reality

Test Types:

- Range validation
- Format validation
- Business rule validation
- Cross-field validation

Examples:

```
-- Test: Age should be between 0 and 120
SELECT COUNT(*) FROM customers
WHERE age < 0 OR age > 120;
-- Expected: 0

-- Test: Order total = sum of line items
SELECT COUNT(*) FROM orders o
WHERE ABS(o.total_amount - (
SELECT SUM(quantity * unit_price)
FROM order_items
WHERE order_id = o.order_id
)) > 0.01;
```

```
-- Expected: 0
```

3. Consistency

Definition: Data is consistent across systems

Test Types:

- Cross-system reconciliation
- Duplicate detection
- Standardization checks
- Temporal consistency

Examples:

```
-- Test: Revenue in DW matches source system
SELECT
ABS(dw.revenue - source.revenue) as diff
FROM data_warehouse.revenue_summary dw
JOIN source_system.revenue_summary source
ON dw.date = source.date
WHERE ABS(dw.revenue - source.revenue) > 100;
-- Expected: 0 rows

-- Test: No duplicate customer records
SELECT email, COUNT(*) as count
FROM customers
GROUP BY email
HAVING COUNT(*) > 1;
-- Expected: 0 rows
```

4. Validity

Definition: Data conforms to defined formats and rules

Test Types:

- Data type validation
- Format checks (email, phone, etc.)
- Enum/category validation
- Regex pattern matching

Examples:

```
-- Test: Email format validation
SELECT COUNT(*) FROM customers
WHERE email NOT LIKE '%_@__%.__%';
-- Expected: 0

-- Test: Status must be valid enum
SELECT COUNT(*) FROM orders
WHERE status NOT IN ('pending', 'shipped', 'delivered', 'cancelled');
-- Expected: 0

-- Test: Phone number format (US)
SELECT COUNT(*) FROM customers
WHERE phone_number IS NOT NULL
AND phone_number NOT REGEXP '^[0-9]{10}$|^[0-9]{3}-[0-9]{3}-[0-9]{4}$';
-- Expected: 0
```

5. Timeliness

Definition: Data is available when needed

Test Types:

- Freshness checks
- SLA monitoring
- Update frequency validation
- Latency measurement

Examples:

```
-- Test: Data should be less than 24 hours old
SELECT COUNT(*) FROM daily_summary
WHERE last_updated < CURRENT_TIMESTAMP - INTERVAL '24 hours';
-- Expected: 0

-- Test: All yesterday's orders loaded
SELECT DATE(order_date) as date, COUNT(*) as count
FROM orders
WHERE DATE(order_date) = CURRENT_DATE - 1
GROUP BY DATE(order_date);
-- Expected: > 0 (should have orders from yesterday)
```

6. Uniqueness

Definition: No unintended duplicates

Test Types:

- Primary key uniqueness
- Business key uniqueness
- Composite key uniqueness
- Duplicate record detection

Examples:

```
-- Test: Order ID is unique
SELECT order_id, COUNT(*) as count
FROM orders
GROUP BY order_id
HAVING COUNT(*) > 1;
-- Expected: 0 rows

-- Test: Customer email + signup_date is unique
SELECT email, signup_date, COUNT(*) as count
FROM customers
GROUP BY email, signup_date
HAVING COUNT(*) > 1;
-- Expected: 0 rows
```

7. Integrity

Definition: Relationships between data are maintained

Test Types:

- Foreign key validation
- Referential integrity
- Parent-child relationships
- Orphan record detection

Examples:

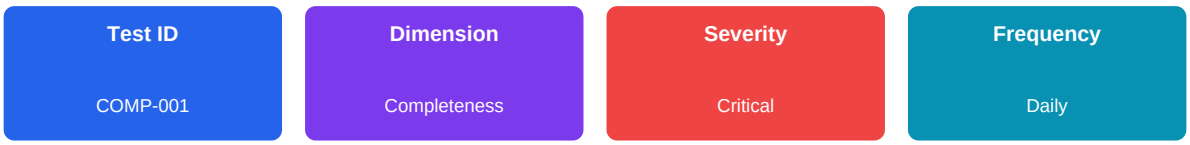
```
-- Test: All orders have valid customer_id
SELECT COUNT(*) FROM orders o
WHERE NOT EXISTS (
SELECT 1 FROM customers c
WHERE c.customer_id = o.customer_id
```

```
);
-- Expected: 0

-- Test: No orphan order items
SELECT COUNT(*) FROM order_items oi
WHERE NOT EXISTS (
SELECT 1 FROM orders o
WHERE o.order_id = oi.order_id
);
-- Expected: 0
```

Test Case Templates

Test Case Structure



Template Structure

Every test case should include:

```
Test ID: Unique identifier (e.g., COMP-001)
Test Name: Descriptive name
Dimension: Quality dimension (Completeness, Accuracy, etc.)
Entity: Table/entity being tested
Description: What is being tested and why
Test Logic: SQL or code to execute test
Expected Result: Pass criteria
Severity: Critical | High | Medium | Low
Frequency: How often to run (hourly, daily, weekly)
Owner: Team/person responsible
```

Example Test Cases

COMP-001: Customer Email Completeness

```
Test ID: COMP-001
Test Name: Customer Email Not Null
Dimension: Completeness
Entity: customers
Description: Email is required for all customers (used for login and communication)
Test Logic:
SELECT COUNT(*) as fail_count
FROM customers
WHERE email IS NULL
Expected Result: fail_count = 0
Severity: Critical
Frequency: Daily (after ETL)
Owner: Data Engineering Team
```

ACC-002: Order Total Accuracy

```
Test ID: ACC-002
Test Name: Order Total Equals Line Items Sum
Dimension: Accuracy
Entity: orders
Description: Order total should equal sum of all line items (quantity * unit_price)
Test Logic:
SELECT
o.order_id,
o.total_amount as order_total,
COALESCE(SUM(oi.quantity * oi.unit_price), 0) as calculated_total,
ABS(o.total_amount - COALESCE(SUM(oi.quantity * oi.unit_price), 0)) as diff
FROM orders o
LEFT JOIN order_items oi ON o.order_id = oi.order_id
GROUP BY o.order_id, o.total_amount
HAVING ABS(o.total_amount - COALESCE(SUM(oi.quantity * oi.unit_price), 0)) > 0.01
Expected Result: 0 rows returned
Severity: Critical
Frequency: Daily (after ETL)
Owner: Data Engineering Team
```

CONS-003: Revenue Reconciliation

```
Test ID: CONS-003
Test Name: DW Revenue Matches Source System
Dimension: Consistency
Entity: revenue_summary
Description: Daily revenue in data warehouse should match source system within $100
Test Logic:
SELECT
dw.date,
dw.revenue as dw_revenue,
src.revenue as source_revenue,
ABS(dw.revenue - src.revenue) as diff
FROM analytics.revenue_summary dw
JOIN source_db.revenue_summary src ON dw.date = src.date
WHERE dw.date >= CURRENT_DATE - 7
AND ABS(dw.revenue - src.revenue) > 100
Expected Result: 0 rows returned
Severity: High
Frequency: Daily
Owner: Analytics Team
```

VAL-004: Email Format Validation

```
Test ID: VAL-004
Test Name: Email Format Valid
Dimension: Validity
Entity: customers
Description: Email should follow standard email format (x@y.z)
Test Logic:
SELECT COUNT(*) as fail_count
FROM customers
WHERE email IS NOT NULL
AND email NOT LIKE '%_@__%.__%'
Expected Result: fail_count = 0
Severity: High
Frequency: Daily
Owner: Data Engineering Team
```

TIME-005: Data Freshness

```
Test ID: TIME-005
Test Name: Orders Table Freshness
Dimension: Timeliness
Entity: orders
Description: Orders table should be updated within last 2 hours
Test Logic:
SELECT
MAX(updated_at) as last_update,
EXTRACT(EPOCH FROM (CURRENT_TIMESTAMP - MAX(updated_at)))/3600 as hours_since_update
FROM orders
WHERE EXTRACT(EPOCH FROM (CURRENT_TIMESTAMP - MAX(updated_at)))/3600 > 2
Expected Result: 0 rows returned
Severity: Medium
Frequency: Hourly
Owner: Data Engineering Team
```

100+ Test Case Library

Completeness — 15 tests

- Required fields not null
- Referential completeness
- Audit fields populated
- Incremental load completeness
- Column population rate
- Minimum row count
- Cascading deletes validated
- Metadata fields present
- Full refresh completeness
- Record count vs source
- Expected columns exist
- Soft delete flags consistent
- Historical records present
- Partition completeness
- Time series continuity

Accuracy — 20 tests

- Numeric range validation
- Percentage bounds (0-100)
- Running totals accuracy
- Rounding validation
- Discount calculation
- Statistical outlier detection
- Control total reconciliation
- Date range validation
- Calculated field accuracy
- Cross-field math validation
- Currency conversion
- Commission calculation
- Known value validation
- Checksum validation
- Future date check
- Aggregation accuracy
- Precision checks
- Tax calculation accuracy
- Metric formulas validated
- Sample record validation

Consistency — 15 tests

- Cross-system reconciliation
- Naming convention consistency
- Currency consistency
- Version consistency
- Temporal consistency
- Duplicate detection
- Unit consistency
- Granularity consistency
- Status transitions valid
- Aggregation consistency
- Case sensitivity consistency
- Time zone consistency
- SCD Type 2 consistency
- Parent-child consistency
- Standardization checks

Validity — 20 tests

- Data type validation
- URL format
- Credit card format
- Enum validation
- JSON validation
- String length limits
- Mime type validation
- Email format
- Zip code format
- IP address format
- Category validation
- XML validation
- Character set validation
- Schema validation
- Phone format
- SSN format
- UUID format
- Boolean validation
- Regular expression patterns
- File extension validation

Timeliness — 10 tests

- Data freshness
- Lag time measurement
- Batch completion time
- Refresh schedule adherence
- SLA adherence
- Pipeline duration
- Historical load timeliness
- Update frequency
- Real-time stream delay
- Incremental load frequency

Uniqueness — 10 tests

- Primary key uniqueness
- Email uniqueness
- Transaction ID uniqueness
- Fuzzy duplicate detection
- Business key uniqueness
- Username uniqueness
- Sequence validation
- Composite key uniqueness
- External ID uniqueness
- UUID uniqueness

Integrity — 15 tests

- Foreign key validation
- Parent record exists
- Hierarchical integrity
- Circular reference detection
- Temporal integrity
- Referential integrity
- Child record count
- Graph relationship validation
- Self-referential integrity
- Version chain integrity
- Orphan record detection
- Many-to-many link validation
- Cascade delete validation
- Cross-database integrity
- Audit trail integrity

Python Automation Scripts

Script 1: Generic SQL Test Runner

Script 2: Pandas DataFrame Validator

Script 3: dbt Test Generator

dbt Integration

dbt Generic Tests

schema.yml:

dbt Custom Tests

tests/orders_total_accuracy.sql:

```
-- Test that order total equals sum of line items
SELECT
o.order_id,
o.total_amount as order_total,
COALESCE(SUM(oi.quantity * oi.unit_price), 0) as calculated_total,
ABS(o.total_amount - COALESCE(SUM(oi.quantity * oi.unit_price), 0)) as diff
FROM {{ ref('orders') }} o
LEFT JOIN {{ ref('order_items') }} oi ON o.order_id = oi.order_id
GROUP BY o.order_id, o.total_amount
HAVING ABS(o.total_amount - COALESCE(SUM(oi.quantity * oi.unit_price), 0)) > 0.01
```

tests/data_freshness.sql:

```
-- Test that data was updated within last 24 hours
SELECT COUNT(*) as stale_records
FROM {{ ref('orders') }}
WHERE last_updated < CURRENT_TIMESTAMP - INTERVAL '24 hours'
HAVING COUNT(*) > 0
```

Great Expectations Integration

Suite Configuration

great_expectations.yml:

Python Example

```
# great_expectations_runner.py
import great_expectations as ge

# Load data
df = ge.read_csv('customers.csv')

# Define expectations
df.expect_column_values_to_not_be_null('customer_id')
df.expect_column_values_to_be_unique('customer_id')
df.expect_column_values_to_match_regex('email', r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$')
df.expect_column_values_to_be_in_set('status', ['active', 'inactive', 'suspended'])
df.expect_column_values_to_be_between('age', min_value=0, max_value=120)

# Validate
results = df.validate()

# Print results
print(f"Success: {results['success']}")
print(f"Passed: {results['statistics']['successful_expectations']}")
print(f"Failed: {results['statistics']['unsuccessful_expectations']}")
```

CI/CD Integration Patterns

CI/CD Pipeline Integration

GitHub Actions

.github/workflows/data-quality-tests.yml:

GitLab CI

.gitlab-ci.yml:

```
stages:
- test
- report

data_quality_tests:
stage: test
image: python:3.9
script:
- pip install -r requirements.txt
- python test_runner.py
artifacts:
paths:
- test_results.json
reports:
junit: test_results.xml
only:
- main
- merge_requests

generate_report:
stage: report
script:
- python generate_report.py test_results.json
artifacts:
paths:
- test_report.html
only:
- main
```

Test Results Dashboard

Dashboard SQL Queries

Test execution history:

Tableau Dashboard Layout

Excel Templates

Template 1: Test Case Inventory

Columns

- Test ID
- Test Name
- Dimension
- Entity
- Description
- Test Logic (SQL)
- Expected Result
- Severity
- Frequency
- Owner
- Status
- Last Run Date
- Last Result
- Notes

Features

- Conditional formatting by severity
- Pivot tables (dimension, entity, owner)
- Auto-filters on all columns
- Data validation dropdowns

Template 2: Test Results Log

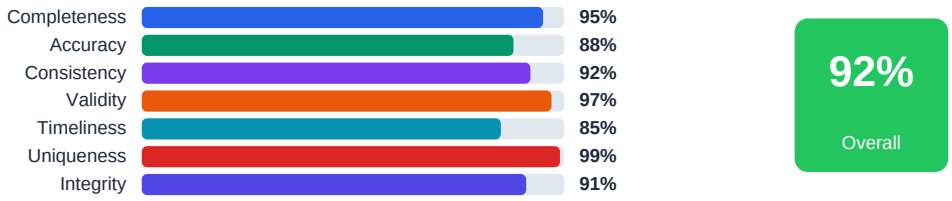
Columns

- Test Run ID
- Test ID
- Test Name
- Execution Timestamp
- Passed (TRUE/FALSE)
- Fail Count
- Error Message
- Execution Time (seconds)

Features

- Automatic date filtering
- Pass/Fail conditional formatting
- Summary statistics
- Charts (pass rate, failures)

Template 3: Data Quality Scorecard



Document Version: 1.0

Last Updated: Jan 2026

Package Contents:

1. This framework document (PDF)
2. Excel templates:

• test_case_inventory.xlsx

• test_results_log.xlsx

• data_quality_scorecard.xlsx

3. Python scripts:
- test_runner.py
- dataframe_validator.py
- dbt_test_generator.py
- great_expectations_runner.py

License: Free to use and distribute

For questions or feedback, visit: https://www.EnterprisedDataSolutions.co.nz

---