UnCodemy

Test Data Management Guide PDF

Test data management is the planning, creation, protection and upkeep of data used in testing. Use realistic but safe data, mask or generate personal information instead of copying production, keep data repeatable with scripts or API setup, version it with tests and clean up after runs so tests stay independent.

Key takeaways

- Never use unmasked production personal data in test environments.
- Good test data covers valid, invalid, boundary and special cases.
- Automated tests should create their own data and clean up afterwards.
- Refresh and reset environments on a schedule to keep data predictable.
- Document where data comes from and who owns it.

What is test data management?

Test data management is the process of providing suitable, consistent and secure data for testing activities, including its creation, masking, refresh, storage and disposal.

Why does test data need management?

Tests are only as reliable as their data. A test that expects a certain customer to exist fails when someone deletes that record. Shared data causes tests to interfere with each other. Poor data hides defects, for example when every sample name is short and ASCII, so long or accented names never get tested. Managing data turns random failures into predictable results.

What kinds of data do tests need?

Kind	Examples
Valid data	Correct emails, normal quantities
Invalid data	Wrong formats, empty values, too long strings
Boundary data	Minimum and maximum values, edge dates
Special data	Unicode, apostrophes, very long names, leap days

Kind	Examples
Volume data	Large tables for performance checks
Reference data	Countries, currencies, product categories
State data	Orders in each status, accounts of each type
Plan the data together with the test cases, since each case implies particular inputs and starting state.	

How can I create test data?

Options include manual entry for small needs, scripts that insert rows, API calls that create records through the normal path, data generation tools that produce realistic fake values, and copies of production data after masking. API creation is attractive because it respects business rules and works for automated tests.

How do I protect personal data?

Do not copy raw production data to lower environments. Use **masking** to replace names, emails, phone numbers and identifiers with realistic fake values while keeping formats, **subsetting** to copy only the needed slice, or **synthetic data** generated from rules. Follow your organisation's privacy policy and local regulations, and restrict access to test environments.

How should automated tests handle data?

Each test creates what it needs with unique values, such as a random suffix on an email, and removes it at the end. Avoid relying on records that were set up by hand. For read-only reference data, use a stable seed that is loaded before the run. Put data builders or factories in shared helper code so every test creates objects the same way.

How do I keep environments predictable?

Reset the database from a known snapshot before major runs, or restore it nightly. Use separate schemas or tenants for parallel work. Track which tests change global settings, and revert them. Monitor data growth, since very large test databases slow down runs.

What is the small example of a data builder?

```
function newCustomer(overrides):
    base = { name: "Test " + randomSuffix(),
             email: randomSuffix() + "@example.test",
             country: "IN" }
    return merge(base, overrides)
```

Tests call `newCustomer({ country: "US" })` to change only what matters. This keeps tests readable and avoids copy-pasted data.

How do I keep documentation of data?

Maintain a short catalogue: dataset name, purpose, owner, source, refresh method, masking status and the tests that use it. Store scripts and seed files in version control with the tests. When the data model changes, update seeds in the same change.

What mistakes are common?

Using one shared user for every test, copying production without masking, leaving old test data to grow forever, hard-coding IDs, ignoring data dependencies between systems and not testing with unusual characters or large values.

How do I plan test data for a new feature?

Read the acceptance criteria and list every input, state and rule. For a discount feature, you may need customers of different types, products at different prices, expired and valid coupons and orders at the minimum and maximum limits. Mark which data is read only and which is changed by tests. Decide who creates it, how it is reset and which tests depend on it. A half-page data plan written before testing saves hours of confusion later.

How do I deal with data that depends on other systems?

When your application relies on payment, email or inventory services, use sandbox accounts or simulators with predictable responses. Keep a list of special values, such as test card numbers published by the provider, and document them. Be careful when a record must exist in several systems at once; use scripts that create them together and verify consistency before the test starts.

What is a simple checklist for clean test data?

Is it masked or synthetic? Is it unique per test? Is it created by code? Is it cleaned afterwards? Is it versioned? Is it documented? A yes to each question means your tests will be calmer and your reports more trustworthy.

How do I practise?

Build a tiny script that creates ten customers with random values through an API or SQL, run a test that uses one of them and delete them at the end. Repeat in parallel to confirm there are no collisions.

Frequently asked questions

What is test data management?

It is the planning, creation, protection, refresh and disposal of data used for testing so that tests are repeatable and safe.

Can I use production data for testing?

Not directly if it contains personal data. Mask or generate data instead, and follow your privacy policy and local rules.

What is data masking?

Masking replaces sensitive values with realistic fake ones while keeping format and relationships so that data remains useful for tests.

How should automated tests handle data?

Each test should create unique data it needs, avoid depending on other tests and clean up after itself.

Why do tests fail because of data?

Shared or changed records, missing seeds and unclear starting states cause failures unrelated to the feature under test.

Sources and further reading

- ISTQB glossary and resources
- ISTQB glossary
- SQLite documentation

About this PDF

These PDFs are study material for learning and revision. They are not a certificate, a degree or a promise of a job, and technology changes, so check official documentation for the latest details. Spot an error? Write to info@uncodemy.com and the trainer team will review it.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550