|  |
| --- |
| Data Migration Test Approach |

Contents

[1 Effective Software Testing When Migrating Data 1](#_Toc43404052)

[1.1 Data Level Validation Testing: 1](#_Toc43404053)

[1.2 Application Level Validation Testing 1](#_Toc43404054)

[2 Test Approach for Data Migration Testing 2](#_Toc43404055)

[2.1 Data Validation Test Design 2](#_Toc43404056)

[2.2 Test Environments 2](#_Toc43404057)

[2.3 Data Validation Test Runs 2](#_Toc43404058)

[2.4 Reporting Bugs 2](#_Toc43404059)

[2.5 Tips to Create an Effective Data Migration Test Approach 3](#_Toc43404060)

# Effective Software Testing When Migrating Data

In software testing, Data Migration Testing is conducted to compare migrated data with original data to discover any discrepancies when moving data from a legacy database(s) to a new destination database.

These data can be migrated either automatically using a migration tool or by manually extracting data from the source database and inserting the data into the destination database.

Data migration testing comprises of Data Level Validation testing and Application Level Validation testing.

## Data Level Validation Testing:

This type of software testing verifies that data has been migrated from multiple databases to a common database without any discrepancies. The following levels of verifications will be performed during Data Level Validation Testing:

* Level 1: Row counts: Verify the number of records that will be migrated.
* Level 2: Data verification: Verify the accuracy of data of a selected sample from the migrated data.
* Level 3: Entitlement verification: Verify the destination database set up for users and data samples.

## Application Level Validation Testing

In Application Level Validation testing, a software tester verifies the functionality of a sample migration application that was migrated from an old legacy system to the new system. Application Level Validation testing ensures smooth running of the migrated application with the new database using following validations:

* Capture or create sample data in the application before migration.
* After migration, log in to the new application and verify a sample data set.
* After migration, log in to legacy systems and verify the locked/unlocked status of accounts.
* Verify customer support access to all legacy systems, despite the user being blocked during the migration process.
* Verify whether new users are prohibited from creating a new account in a legacy system after launching the new application.
* Verify immediate reinstatement of user access to the legacy system if migration to the new system fails.
* Verify the termination of access to legacy systems at migration.
* Validate system login credentials for the new application.

# Test Approach for Data Migration Testing

## Data Validation Test Design

When you test database migration, it is important to create a set of SQL queries to validate the data before (source database) and after (destination database) migration. The validation queries can be arranged in a hierarchy and it should cover the designed scope.

For example, to test if all users have been migrated, it is essential to check how many users are in the source database and how many have been migrated. Checking the row counts of each database will ensure this.

Take a sample data set from the source and compare the data with the destination data in the database.

Testing tips:

* Always take the most important dataset and data that have values in all columns
* Verify the data types after the migration
* Check the different time formats/zones, currencies etc.
* Check for data with special characters
* Find data that should not be migrated
* Check for duplicated data after migration
* A sample test case can be created as shown below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **#** | **Test Description** | **Input Data** | **Source Query (Before Migration)** | **Destination Query (After Migration)** | **Expected Results** | **Test Results** |
| 1 | Verify total count of migrated users | Not applicable | SELECT COUNT (\*) FROM Users | SELECT COUNT (\*) FROM Reg\_Users | Raw count should be same before and after the data migration. | Not Executed |
| 2 | Validate data of 5 rows in the user table | 5 rows | SELECT \* FROM Users WHERE ROWNUM <=5; | SELECT \* FROM Reg\_Users WHERE ROWNUM <=5; | Data in the selected table rows should be identical before and after the migration. | Not Executed |

## Test Environments

Test environments should consist of a source database copy and a blank isolated destination database. A tester can migrate data using a migration tool, which will facilitate the migration both table-by-table and using a set of reference tables. The tool should be able to accommodate a large data load since the data can include data from historical databases.

## Data Validation Test Runs

The database migration process must be completed prior to the test depending on the test design.

## Reporting Bugs

If the migration test fails, it is important to report the bug with the following information:

* Number of failed rows and columns
* Name of the failed object
* Database error logs
* The query used to validate the data
* User account information used to run the validation
* Date and time of the test
* Semantic errors

## Tips to Create an Effective Data Migration Test Approach

* Users should be able to access existing data and post migration data easily without any issues.
* Performance of database should be the same or better after the migration.
* Note the duration of the migration. A duration of a migration can be long, running the risk of application downtime.
* Have a copy of the source databases to conduct re-tests at any time on a new database. It will also help reproduce the bugs.
* Corrupted data should not be migrated to the destination database, and necessary actions should be taken to resolve the corrupted data.
* Get stakeholders involved the migration plan, as their permission to access different data sources could be mandatory.
* Make sure there are no inconsistencies in currency, date and time, time zone fields, and decimal points of currencies.

Few Sample SQL Statements recommended to run on Source and destination databases:

* Total # of Table count in QA
* Number of attributes for all tables
* Table Names, attributes, type, size, and nullable.
* Number of views in QA DB
* Number of User profiles in QA DB
* Number of Active User profiles in QA DB
* Number of Inactive User profiles in QA DB
* Number of orders in QA DB
* Number of orders with status "Ready for Delivery" in QA DB
* Number of orders with status "Available" in QA DB
* Number of orders with status "On Hold" in QA DB
* Number of orders with status "Sold Remainder Canceled" in QA DB
* Number of orders with status "In Process" in QA DB
* Number of orders with status "On Order" in QA DB
* Number of orders with status "Partially Available" in QA DB
* Number of orders with status "Ready for Pickup" in QA DB
* Number of orders with status "Sold" in QA DB
* Number of orders with status "Canceled" in QA DB
* Number of Accounts in QA DB
* Number of Active Accounts in QA DB
* Number of Inactive Accounts in QA DB
* Number of Parent Accounts in QA DB
* Number of child Accounts in QA DB
* Number of child Accounts under each single parent account in QA DB
* Number of Categories in QA DB
* Number of Active Categories in QA DB
* Number of Inactive Categories in QA DB
* Number of SKUs in QA DB
* Number of inactive SKUs in QA DB
* Number of Discontinued SKU’s in QA DB
* Number of Shipping address in QA DB
* Number of Billing address in QA DB
* Number of Shipping and Billing address in QA DB
* Number of Null address in QA DB