**What is Database Testing?**

**Database Testing** is a type of software testing that checks the schema, tables, triggers, etc. of the Database under test. It also checks data integrity and consistency. It may involve creating complex queries to load/stress test the Database and check its responsiveness.

## Why Database Testing is Important?

**Database Testing is Important** in software testing because it ensures data values and information received and stored into database are valid or not. Database testing helps to save data loss, saves aborted transaction data and no unauthorized access to the information. Database is important for any software application hence testers must have good knowledge of SQL for database testing.

The GUI is usually given the most emphasis by the test and development team members since the Graphical User Interface happens to be the most visible part of the application. However, what is also important is to validate the information that is the heart of the application, aka DATABASE.

Let us consider a Banking application wherein a user makes transactions. Now from Database Testing or DB Testing viewpoint following, things are important:

1. The application stores the transaction information in the application database and displays them correctly to the user.
2. No information is lost in the process.
3. No partially performed or aborted operation information is saved by the application.
4. No unauthorized individual is allowed to access the user’s information.

To ensure all these above objectives, we need to use data validation or data testing.

## Differences between User-Interface Testing and Data Testing

| **User-Interface testing** | **Database or Data testing** |
| --- | --- |
| This type of testing is also known as Graphical User Interface testing or Front-end Testing. | This type of testing is also known as Backend Testing or data testing. |
| This type of testing chiefly deals with all the testable items that are open to the user for viewership and interaction like Forms, Presentation, Graphs, Menus, and Reports, etc. (created through VB, VB.net, VC++, Delphi – Front-end Tools ) | This type of testing chiefly deals with all the testable items that are generally hidden from the user for viewership. These include internal processes and storage like Assembly, DBMS like Oracle, SQL Server, MYSQL, etc. |
| This type of testing includes validating the   * text boxes * select dropdowns * calendars and buttons * Page navigation * display of images * Look and feel of the overall application | This type of testing involves validating:   * the schema * database tables * columns * keys and indexes * stored procedures triggers * database server validations * validating data duplication |
| The tester must be thoroughly knowledgeable about the business requirements as well as the usage of the development tools and the usage of automation frameworks and tools. | To be able to perform backend testing, must the tester have a strong background in the database server and Structured Query Language concepts. |

## ****Types of Database Testing****

The 3 types of Database Testing are

1. Structural Testing
2. Functional Testing
3. Non-functional Testing

## Structural Database Testing

**Structural Database Testing** is a database testing technique that validates all the elements inside data repository that are mainly used for data storage and which are not allowed to be directly manipulated by end-users. The validation of database servers is also an important consideration in structural database testing. A successful completion of this testing needs mastery in SQL queries.

## What is Schema Testing?

**Schema Testing** in database testing validates various schema formats associated with the database and verifies whether the mapping formats of tables/views/columns are compatible with mapping formats of user interface. The main purpose of schema testing is to ensure the schema mapping between front-end and back-end are similar. Thus, it is also referred to as **mapping testing.**

Let us discuss the most important checkpoints for schema testing.

1. Validation of the various schema formats associated with the databases. Many times the mapping format of the table may not be compatible with the mapping format present in the user interface level of the application.
2. There is a need for verification in the case of unmapped tables/views/columns.
3. There is also a need to verify whether heterogeneous databases in an environment are consistent with the overall application mapping.

Let us also look at some of the interesting Database Testing tools for validating database schemas.

* DBUnit that is integrated with Ant is very suitable for mapping testing.
* SQL Server allows the testers to be able to check and to query the schema of the Database by writing simple queries and not through code.

For example, if the developers want to change a table structure or delete it, the tester would want to ensure that all the Stored Procedures and Views that use that table are compatible with the particular change. Another example could be that if the testers want to check for schema changes between 2 databases, they can do that by using simple queries.

## Database Table, Column Testing

Let us look into various checks for database and column testing.

1. Whether the mapping of the database fields and columns in the backend is compatible with those mappings in the front-end?
2. Validation of the length and naming convention of the database fields and columns as specified by the requirements.
3. Validation of the presence of any unused/unmapped database tables/columns.
4. Validation of the compatibility of the

* datatype
* field lengths

of the back-end database columns with that of those present at the front-end of the application.

1. Whether the database fields allow the user to provide desired user inputs as required by the business requirement specification documents.

## Keys and indexes testing

Important checks for keys and indexes –

1. Check whether the required

* Primary Key
* Foreign Key

constraints have been created on the required tables.

1. Check whether the references for foreign keys are valid.
2. Check whether the data type of the primary key and the corresponding foreign keys are the same in the two tables.
3. Check whether the required naming conventions have been followed for all the keys and indexes.
4. Check the size and length of the required fields and indexes.
5. Whether the required

* Clustered indexes
* Non Clustered indexes

have been created on the required tables as specified by the business requirements.

## Stored Procedures Testing

Important tests to check stored procedures are:

1. Whether the development team did adopt the required

* coding standard conventions
* exception and error handling

for all the stored procedures for all the modules for the application under test.

1. Whether the development team did cover all the conditions/loops by applying the required input data to the application under test?
2. Whether the development team did properly apply the TRIM operation whenever data is fetched from the required tables in the Database?
3. Whether the manual execution of the Stored Procedure provides the end-user with the required result?
4. Whether the manual execution of the Stored Procedure ensures the table fields are being updated as required by the application under test?
5. Whether the execution of the Stored Procedures enables the implicit invoking of the required triggers?
6. Validation of the presence of any unused stored procedures.
7. Validation for Allow Null condition which can be done at the database level.
8. Validation of the fact that all the Stored Procedures and Functions have been successfully executed when the Database under test is blank.
9. Validation of the overall integration of the stored procedure modules as per as the requirements of the application under test.

Some of the useful Database Testing tools for testing stored procedures are LINQ , SP Test tool etc.

## Trigger Testing

1. Whether the required coding conventions have been followed during the coding phase of the Triggers?
2. Check whether the triggers executed for the respective DML transactions have fulfilled the required conditions.
3. Whether the trigger updates the data correctly once they have been executed?
4. Validation of the required Update/Insert/Delete triggers functionality in the realm of the application under test.

## Database Server Validations

1. Check the database server configurations as specified by the business requirements.
2. Check the authorization of the required user to perform only those levels of actions that are required by the application.
3. Check that the database server is able to cater to the needs of the maximum allowed number of user-transactions as specified by the business requirement specifications.

## Functional Database Testing

**Functional Database Testing** is a type of database testing that is used to validate the functional requirements of a database from the end-user’s perspective. The main goal of functional database testing is to test whether the transactions and operations performed by the end-users which are related to the database works as expected or not.

Following are the basic conditions that need to be observed for database validations.

* Whether the field is mandatory while allowing NULL values on that field?
* Whether the length of each field is of sufficient size?
* Whether all similar fields have the same names across tables?
* Whether there are any computed fields present in the Database?

This particular process is the validation of the field mappings from the end-user viewpoint. In this particular scenario, the tester would perform an operation at the database level and then would navigate to the relevant user interface item to observe and validate whether the proper field validations have been carried out or not.

The vice versa condition whereby, first operation is carried out by the tester at the user interface, and then the same is validated from the back end is should also be done.

## Checking data integrity and consistency

Following checks are important

1. Whether the data is logically well organized?
2. Whether the data stored in the tables is correct and as per the business requirements?
3. Whether there are any unnecessary data present in the application under test?
4. Whether the data has been stored as per as the requirement with respect to data which has been updated from the user interface?
5. Whether the TRIM operations performed on the data before inserting data into the Database under test?
6. Whether the transactions have been performed according to the business requirement specifications and whether the results are correct or not?
7. Whether the data has been properly committed if the transaction has been successfully executed?
8. Whether the data has been rolled backed successfully if the transaction has not been executed successfully by the end-user?
9. Whether the data has been rolled backed if the transaction has not been executed successfully and multiple heterogeneous databases have been involved in the transaction in question?
10. Whether all the transactions have been executed by using the required design procedures as specified by the system business requirements?

## Login and User Security

The validations of the login and user security credentials need to take into consideration the following things.

1. Whether the application prevents the user from proceeding further in the application in case of a

* invalid username but valid password
* valid username but invalid password.
* invalid username and invalid password.

1. Whether the user is allowed to perform only those specific operations which are specified by the business requirements?
2. Whether the data is secured from unauthorized access?
3. Whether there are different user roles created with different permissions?
4. Whether all the users have required levels of access on the specified Database as required by the business specifications?
5. Check that sensitive data like passwords, creditcard numbers are encrypted and not stored as plain text in Database. It is a good practice to ensure all accounts should have passwords that are complex and not easily guessed.

## Non-functional testing

[Non-functional testing](https://www.guru99.com/non-functional-testing.html) in the context of database testing can be categorized into various categories as required by the business requirements. These can be load testing, [Stress Testing](https://www.guru99.com/stress-testing-tutorial.html), [Security Testing](https://www.guru99.com/what-is-security-testing.html), [Usability Testing](https://www.guru99.com/usability-testing-tutorial.html), and [Compatibility Testing](https://www.guru99.com/compatibility-testing.html), and so on. The load testing, as well as stress testing, which can be grouped under the gamut of [Performance Testing](https://www.guru99.com/performance-testing.html) serves two specific purposes when it comes to the role of non-functional testing.

**Risk quantification**– Quantification of risk helps the stakeholders to ascertain the various system response time requirements under required levels of load. This is the original intent of any [quality assurance](https://www.guru99.com/all-about-quality-assurance.html) task. We need to note that load testing does not mitigate risk directly, but through the processes of risk identification and risk quantification, presents corrective opportunities and an impetus for remediation that will mitigate risk.

**Minimum system equipment requirement**– The minimum system configuration that will allow the system to meet the formally stated performance expectations of stakeholders. So that extraneous hardware, software, and the associated cost of ownership can be minimized. This particular requirement can be categorized as the overall business optimization requirement.

## ****Load Testing****

The purpose of any load test should be clearly understood and documented. The following types of configurations are a must for load testing.

1. The most frequently used user transactions have the potential to impact the performance of all of the other transactions if they are not efficient.
2. At least one non-editing user transaction should be included in the final test suite, so that performance of such transactions can be differentiated from other more complex transactions.
3. The more important transactions that facilitate the core objectives of the system should be included, as failure under a load of these transactions has, by definition, the greatest impact.
4. At least one editable transaction should be included so that performance of such transactions can be differentiated from other transactions.
5. Optimum response time under huge number of virtual users for all the prospective requirements.
6. Effective times for fetching of various records.