**What is Database Testing?**

Database Testing is checking the schema, tables, triggers, etc. of the database under test. It may involve creating complex queries to load/stress test the database and check its responsiveness. It Checks data integrity and consistency.

|  |
| --- |
| **Database or Data testing** |
| This type of testing is also known as Back-end Testing or data testing. |
| This type of testing chiefly deals with all the testable items that are generally hidden from the user for viewership. These include internal process and storage like Assembly, DBMS like Oracle, [SQL](https://www.guru99.com/sql.html) Server, MYSQL, etc. |
| 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 in order to be able to perform back-end testing must have a strong background in the database server and Structured Query Language concepts. |

The 3 types of Database Testing are

### Structural database testing

The structural data testing involves the validation of all those elements inside the data repository that are used primarily for storage of data and which are not allowed to be directly manipulated by the end users. The validation of the database servers is also a very important consideration in these types of testing. The successful completion of this phase by the testers involves mastery in SQL queries.

**Schema testing -** The chief aspect of schema testing is to ensure that the schema mapping between the front end and back end are similar. Thus, we may also refer to schema testing as **mapping testing.**

Let us discuss 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 the need for verification in the case 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 tools for validating database schemas.

* **DBUnit** that is integrated with Ant is a 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 back end 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

* data type
* field lengths of the backend database columns with that of those present in 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 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 -** The list of the most important things which are to be validated for the stored procedures.

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 operations 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 forAllow 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 interesting 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 which are required by the application.
3. Check that the database server is able to cater to the needs of maximum allowed number of user transactions as specified by the business requirement specifications.

Functional database testing

The Functional database testing as specified by the requirement specification needs to ensure most of those transactions and operations as performed by the end users are consistent with the requirement specifications.

Following are the basic conditions which 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 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 data base 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 an operation is carried out by the tester at the user interface and then the same is validated from the back end is also considered to be a valid option.

**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 as per the business requirements.
8. Whether the data has been roll backed successfully if the transaction has not been executed successfully by the end user.
9. Whether the data has been roll backed at all in the condition that 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 to proceed further in the application in case of a

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

1. Whether the user is allowed to perform only those specific operations which are specified by the business requirements.
2. Whether the data 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, credit card 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

Nonfunctional testing 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 nonfunctional testing.

**Risk quantification**- Quantification of risk actually 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 task. We need to note that load testing does not mitigate risk directly, but through the processes of risk identification and of risk quantification, presents corrective opportunities and an impetus for remediation that will mitigate risk.

**Minimum system equipment requirement**- The understanding which we observe through formal testing, the minimum system configuration that will allow the system to meet the formal 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 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. The observation of the optimum response time under huge number of virtual users for all the prospective requirements.
6. The observation of the effective times for fetching of various records.

Important load testing tools are load runner, win runner and JMeter.

**Stress testing**

Stress testing is also sometimes referred to as torturous testing as it stresses the application under test with enormous loads of work such that the system fails .This helps in identifying breakdown points of the system.

Important stress testing tools are load runner, win runner and JMeter.

Most common occurring issues during database testing

1. Significant amount of overhead could be involved in order to determine the state of the database transactions.
2. Solution: The overall process planning and timing should be organized so that no time and cost based issues appear.
3. New test data have to be designed after cleaning up of the old test data.
4. Solution: A prior plan and methodology for test data generation should be at hand.
5. An SQL generator is required to transform SQL validators in order to ensure the SQL queries are apt for handling the required database test cases.
6. Solution: Maintenance of the SQL queries and their continuous updating is a significant part of the overall testing process which should be part of the overall test strategy.
7. The above mentioned prerequisite ensure that the set-up of the database testing procedure could be costly as well as time consuming.
8. Solution: There should be a fine balance between quality and overall project schedule duration.

**Myths or Misconceptions related to Database Testing.**

1. Database Testing requires plenty of expertise and it is a very tedious job

* Reality: Effective and efficient Database testing provides long-term functional stability to the overall application thus it is necessary to put in hard work behind it.

1. Database testing adds extra work bottleneck

* Reality: On the contrary, database testing adds more value to the overall work by finding out hidden issues and thus pro-actively helping to improve the overall application.

1. Database testing slows down the overall development process

* Reality: Significant amount of database testing helps in the overall improvement of quality for the database application.

1. Database testing could be excessively costly

* Reality: Any expenditure on database testing is a long-term investment which leads to long-term stability and robustness of the application. Thus expenditure on database testing is necessary.

**Best Practices**

* All data including the metadata as well as the functional data needs to be validated according to their mapping by the requirement specification documents.
* Verification of the test data which has been created by / in consultation with the development team needs to be validated.
* Validation of the output data by using both manual as well as automation procedures.
* Deployment of various techniques such as the cause effect graphing technique, equivalence partitioning technique and boundary-value analysis technique for generation of required test data conditions.
* The validation rules of referential integrity for the required database tables also need to be validated.
* The selection of default table values for validation on database consistency is a very important concept Whether the log events have been successfully added in the database for all required login events
* Does scheduled jobs execute in timely manner?
* Take timely backup of Database.