Database Testing

The next step in securing our website is maintaining a safe and efficient database. Let's find out how.

WE'LL COVER THE FOLLOWING ^

Apart from testing our user interface and API functionality, we need to make sure our databases are working properly.

Databases are fundamental to our website as the communication between us and the user depends on data manipulation on the databases. Users retrieve and store data through the user interface. Hence, frontend testing for the UI is pointless if the backend database doesn't work properly.

As we've learned, there are two types of databases, relational (Oracle, SQL) and non-relational (MongoDB, CouchDB) databases, both of which require different approaches for testing. Visit the Web Development course for a quick refresher on the definitions of the two database systems.

With that in my mind, let's talk about testing the conventional relational databases.

Relational Databases

A relational or SQL database should be tested for the following properties:

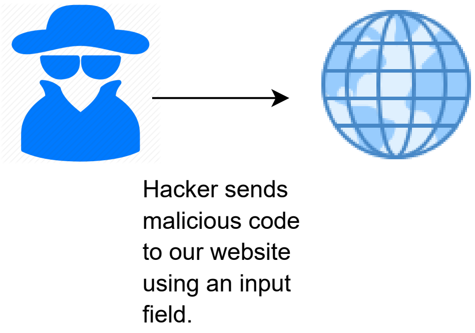
- The database must follow the ACID and CRUD principles.
- Our database must appropriately store and retrieve data. Queries must consistently return the correct results across all users. **TOAD** and

phpMyAdmin are very convenient automated query tools.

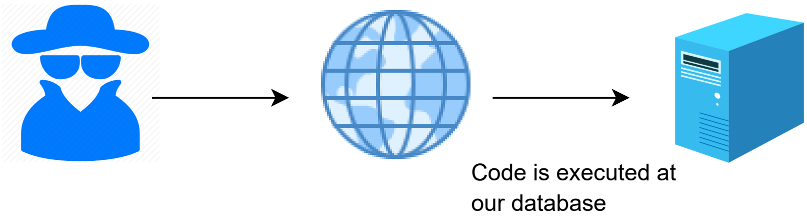
- The fields in our database must correctly match the fields in the frontend interface. This is called *mapping*. **DBUnit** with **Ant** tests database-frontend mapping. Field constraints must be consistent at both ends.
- The database must not collapse under load or stress (several multiple read/write operations). **HammerDB** is a good option to carry out load tests.
- If information in one table is modified, all linked tables must update themselves accordingly. If there is a fault in interconnectivity, the whole site's functionality is at stake.

Security Testing

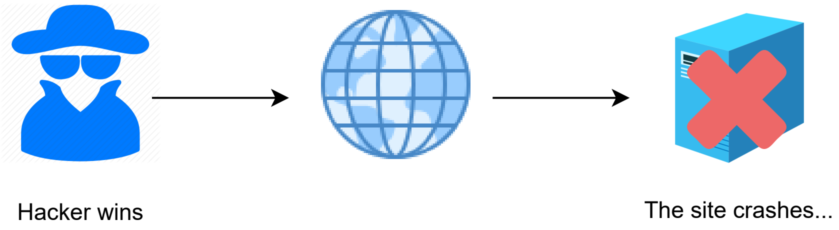
- The DB must be resistant against SQL injections. **Vega** and **Wapiti** are open source tools suitable for SQL injection testing.


Hacker sends malicious code to our website using an input field.

1 of 3

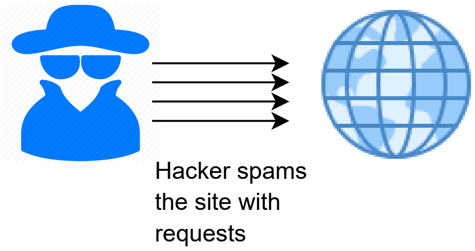


2 of 3

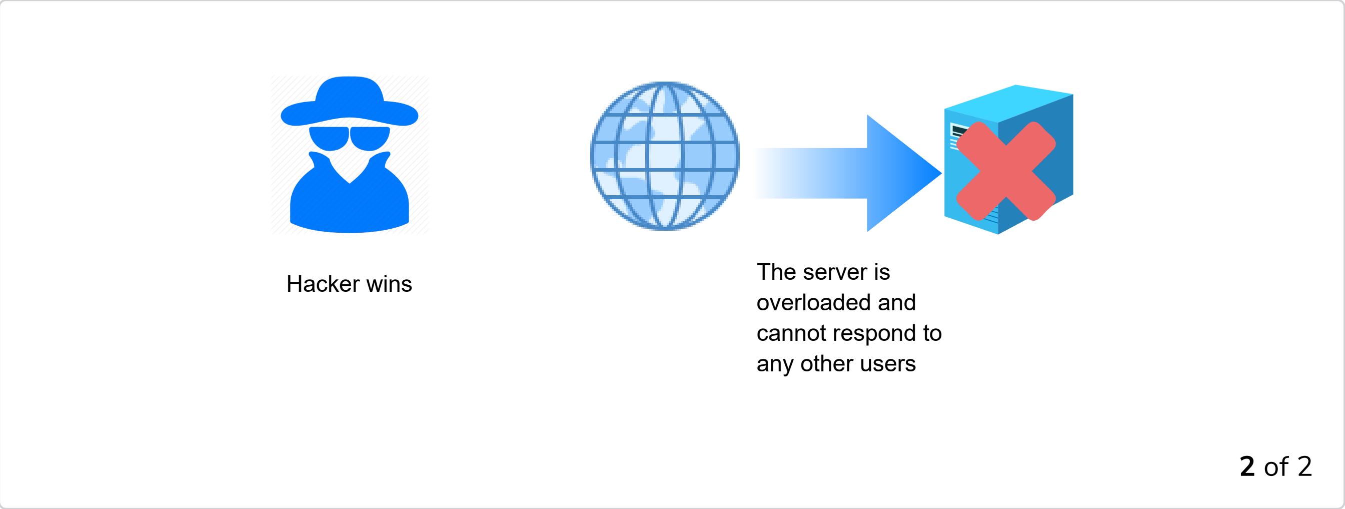


3 of 3

- Unauthorized access is a common attack on SQL databases. This can be avoided by fortifying user authentication checks.
- We must perform penetration testing to check for vulnerabilities in the logic of our DB system. One of the most popular penetration testing tools is **Zed Attack Proxy**.
- Denial of Service attacks can make databases unusable, which is really bad for our website! However, these attacks have to be prevented on the network layer.



1 of 2


Hacker wins
The server is overloaded and cannot respond to any other users
2 of 2

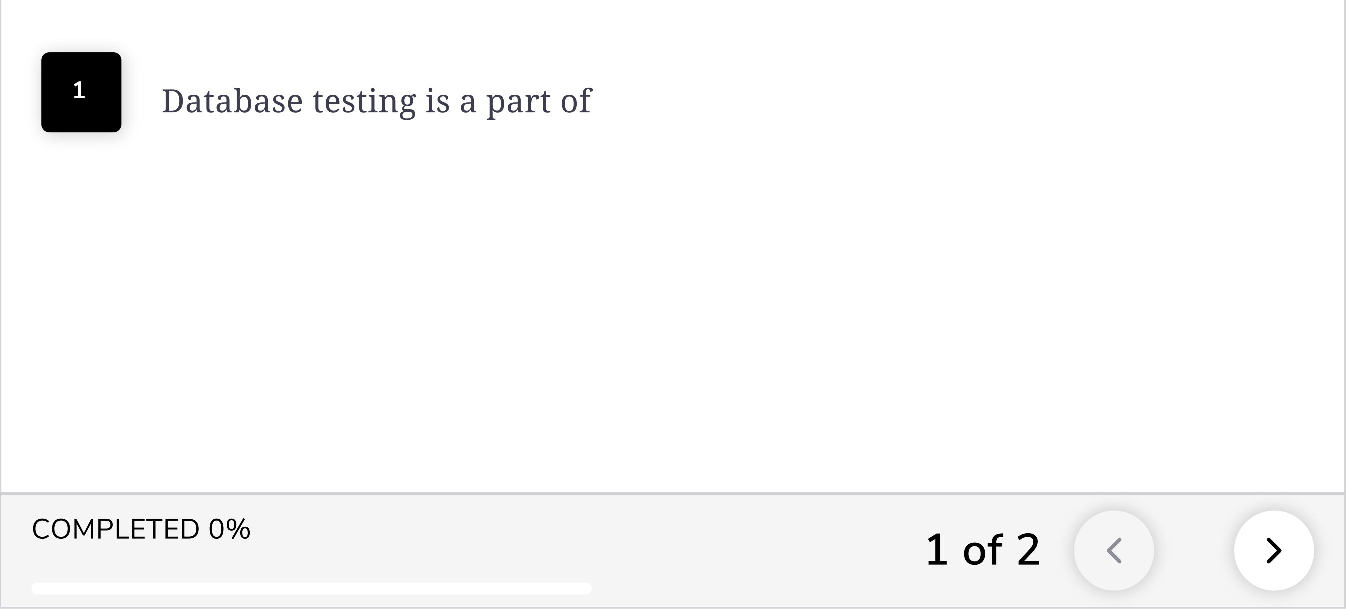
Non-Relational Databases

Non-relational (NoSQL) databases are simpler in terms of structure, and hence, require a lower degree of testing. However, there are still some crucial factors that need to be tested in order to make our website's backend free of flaws.

- The format of the data objects should be consistent. We can check this by performing a few queries. This sort of unit testing can be done using **NoSQLUnit**, which supports several languages including MongoDB.
- We must test data conversions between the backend and the frontend.
- For continuous integration in databases, **Travis CI** is a wonderful option as it supports several NoSQL languages. Testing is also isolated and does not affect the original state of the application.

Security Testing

- We must make sure that our data objects are encrypted.
- User authentication should be thorough and secure.
- NoSQL databases can also be subject to injection attacks by providing a JSON file in a field (assuming our backend stores JSON files). For more details on JSON injections, check out the examples mentioned here.



1 Database testing is a part of

COMPLETED 0%

1 of 2