**QA Interview Questions**

**1. What is software testing?**

* Explain the process of evaluating a software application to ensure it meets the required specifications and is free of defects.

**2. What are the different levels of testing?**

* Unit testing, integration testing, system testing, and acceptance testing.

**3. What is the difference between functional and non-functional testing?**

* **Functional testing** checks the functionality of the software (e.g., does it do what it’s supposed to?).
* **Non-functional testing** checks performance, usability, reliability, etc. (e.g., how well does the software perform under load?).

**4. What is the difference between verification and validation in software testing?**

* **Verification** ensures the product is built correctly (i.e., checking the process).
* **Validation** ensures the right product is built (i.e., checking the product).

**5. What are test cases and why are they important?**

* A **test case** is a set of conditions or actions used to determine if a software feature is working as expected. It’s important because it provides a structured approach to testing.

**6. What is regression testing?**

* Regression testing involves re-testing the software after changes (such as updates or bug fixes) to ensure that the changes haven’t affected existing functionality.

**7. Explain the difference between manual and automated testing.**

* **Manual testing** involves human testers executing test cases without using automation tools.
* **Automated testing** involves using scripts and testing tools to automatically execute test cases.

**8. What is a bug life cycle?**

* The bug life cycle describes the process a bug goes through from being discovered to being resolved. It typically includes stages such as "New", "Assigned", "In Progress", "Fixed", and "Closed".

**9. What are the types of testing you perform for web applications?**

* Examples: functional testing, usability testing, performance testing, compatibility testing, security testing.

**10. What is performance testing, and how do you conduct it?**

* Performance testing evaluates the speed, responsiveness, and stability of an application under load. It can include load testing, stress testing, and scalability testing.

**11. What is the difference between black-box and white-box testing?**

* **Black-box testing** focuses on testing the software’s functionality without knowledge of the internal code.
* **White-box testing** involves testing the internal structure and logic of the software, typically done by developers.

**12. What is a test plan, and what does it include?**

* A **test plan** is a document that outlines the scope, approach, resources, and schedule for testing activities. It typically includes test objectives, testing strategies, resource requirements, and test deliverables.

**13. What is the importance of defect tracking in testing?**

* Defect tracking helps in identifying, managing, and resolving defects in the software. It ensures that all bugs are tracked and addressed efficiently, improving the overall quality.

**14. What is boundary value analysis?**

* Boundary value analysis is a technique in which test cases are designed to include values at the boundaries of input ranges. It helps in identifying defects that occur at extreme or boundary values.

**15. What is exploratory testing?**

* Exploratory testing is an informal testing approach where testers explore the application, learning about it, and designing tests as they go.

**16. What is smoke testing?**

* **Smoke testing** is a preliminary test to check the basic functionality of the software. It ensures that the build is stable enough for further detailed testing.

**17. What is sanity testing?**

* **Sanity testing** is a type of testing that is conducted to verify that specific functionalities or bug fixes are working after changes, without going into full regression testing.

**18. What are the different types of testing methods?**

* **Static Testing:** Testing that occurs without executing code, such as code reviews and inspections.
* **Dynamic Testing:** Testing that involves executing the code and validating functionality.
* **Manual Testing** vs. **Automated Testing**
* **Alpha Testing** vs. **Beta Testing**

**19. What is the difference between alpha testing and beta testing?**

* **Alpha testing** is performed by the internal team (developers, testers) in a controlled environment.
* **Beta testing** is done by a selected group of external users who provide feedback before the software is released to the public.

**20. What is the purpose of boundary value analysis and equivalence partitioning?**

* **Boundary value analysis (BVA)** focuses on testing the edges of input data, as errors are often found near the boundaries.
* **Equivalence partitioning** divides the input data into equivalent classes, reducing the number of test cases while covering a wide range of possibilities.

**21. What is the role of a test lead in software testing?**

* The **test lead** is responsible for managing the testing team, creating test plans, assigning tasks, ensuring test cases are executed, reporting defects, and ensuring the quality of the final product.

**22. What is the importance of test documentation?**

* Test documentation, such as test plans, test cases, and test reports, ensures that testing is systematic, repeatable, and traceable. It also helps with communication and provides a record for future reference.

**23. What is the difference between a test scenario and a test case?**

* **Test scenario** is a high-level description of what needs to be tested, while a **test case** is a detailed set of conditions, inputs, and steps to be followed to verify the functionality of a specific feature.

**24. What is a risk-based testing approach?**

* **Risk-based testing** prioritizes testing based on the risk of failure and its impact on the application. Critical functionalities that have a higher likelihood of failure or a higher impact are tested first.

**25. What is exploratory testing, and when would you use it?**

* **Exploratory testing** involves simultaneous learning, test design, and test execution. Testers use their creativity to explore the application, usually when there is little or no documentation available.

**26. What is the difference between stress testing and load testing?**

* **Load testing** measures the system's performance under expected load conditions.
* **Stress testing** pushes the system beyond its limits to see how it behaves under extreme conditions (e.g., resource depletion or excessive traffic).

**27. What is the significance of version control in software testing?**

* **Version control** helps keep track of changes made to code, test scripts, or documents, ensuring that testers always have access to the latest version of the software and related resources.

**28. Explain the concept of defect severity and priority.**

* **Severity** refers to the impact or seriousness of a defect (e.g., critical, major, minor).
* **Priority** refers to the urgency of fixing the defect (e.g., high, medium, low).

**29. What is the difference between test-driven development (TDD) and behavior-driven development (BDD)?**

* **TDD (Test-Driven Development)** is a development approach where tests are written before the code, ensuring that the code meets predefined tests.
* **BDD (Behavior-Driven Development)** focuses on writing tests in a more natural language (using Gherkin syntax) to describe the behavior of an application.

**30. What are mock objects in unit testing?**

* **Mock objects** are simulated objects that mimic the behavior of real objects. They are used in unit testing to isolate the class under test and avoid dependencies on other components.

**31. What is the difference between static testing and dynamic testing?**

* **Static testing** refers to reviewing and inspecting the code without executing it (e.g., code reviews, static analysis).
* **Dynamic testing** involves executing the code to validate its behavior (e.g., running test cases).

**32. What is the difference between a defect and a failure?**

* A **defect** is a flaw in the software that could cause an issue, while a **failure** is when the defect actually manifests during testing or in production.

**33. What is the difference between a build and a release?**

* A **build** is a version of the software that has been compiled and is ready for testing, while a **release** is the version of the software that is made available to users or customers.

**34. What is A/B testing?**

* **A/B testing** compares two versions of a webpage, feature, or product to determine which one performs better based on user engagement or other metrics.

**35. What is a "clean" environment in testing?**

* A **clean environment** means that the testing environment is set up fresh and free of any prior configurations or data, ensuring that tests run under controlled and unbiased conditions.

**36. How do you handle situations where there is insufficient documentation for testing?**

* In such cases, you can rely on exploratory testing, work closely with developers and business analysts to understand the application’s functionality, and create your own test cases based on use cases or user stories.

**37. What is the difference between acceptance testing and system testing?**

* **Acceptance testing** verifies if the software meets the business requirements, typically performed by the end-user or client.
* **System testing** is the testing of the entire system to ensure it meets the specified requirements and behaves as expected.

**38. What are some common testing tools you’ve used, and what are their purposes?**

* Examples include Selenium (for automation), JMeter (for performance testing), Postman (for API testing), Jenkins (for continuous integration), and Jira (for bug tracking).

**39. Explain the concept of Continuous Integration (CI) and Continuous Testing (CT).**

* **Continuous Integration (CI)** involves regularly integrating code into a shared repository, which is then automatically tested.
* **Continuous Testing (CT)** ensures that automated tests are continuously run on the codebase to detect defects as early as possible.

**40. How do you determine the exit criteria for a testing project?**

* Exit criteria are conditions that must be met before testing is considered complete. These could include the completion of planned test cases, no high-severity defects, test coverage, and the software meeting acceptance criteria.

**TestNG interview questions:**

**1. What is TestNG?**

* **Answer:** TestNG is a testing framework inspired by JUnit and NUnit but introduces some new functionalities that make it more powerful and easier to use. It is used for unit testing, integration testing, and end-to-end testing in Java applications. TestNG supports annotations, parallel test execution, and configurable test suites.

**2. Explain the basic annotations in TestNG.**

* **Answer:**
  + @Test: Marks a method as a test method.
  + @BeforeMethod: Runs before each test method.
  + @AfterMethod: Runs after each test method.
  + @BeforeClass: Runs once before the first test method in the class.
  + @AfterClass: Runs once after the last test method in the class.
  + @BeforeTest: Runs before any test method in the XML file is run.
  + @AfterTest: Runs after all test methods in the XML file are run.
  + @BeforeSuite: Runs once before the execution of the test suite.
  + @AfterSuite: Runs once after the execution of the test suite.
  + @DataProvider: Provides data to the test method.

**3. How do you define a TestNG test suite?**

* **Answer:** A TestNG test suite is defined in an XML file (usually testng.xml). In this file, you can specify which tests to run, group tests, set dependencies, and configure other test execution settings.

<?xml version="1.0" encoding="UTF-8"?>

<suite name="Test Suite">

<test name="Test 1">

<classes>

<class name="TestClass"/>

</classes>

</test>

</suite>

**4. What are the different types of assertions in TestNG?**

* **Answer:**
  + assertEquals(actual, expected): Asserts that two values are equal.
  + assertTrue(condition): Asserts that the condition is true.
  + assertFalse(condition): Asserts that the condition is false.
  + assertNull(object): Asserts that the object is null.
  + assertNotNull(object): Asserts that the object is not null.

**5. What is the use of @DataProvider annotation in TestNG?**

* **Answer:** The @DataProvider annotation is used to provide a set of data to a test method. It allows you to run the same test with different input values. The data provider returns a two-dimensional array or an Iterator<Object[]>, and the test method consumes this data to execute multiple iterations.

@DataProvider(name = "testData")

public Object[][] createData() {

return new Object[][] { {1, 2}, {3, 4} };

}

@Test(dataProvider = "testData")

public void testMethod(int a, int b) {

// Test code using a and b

}

**6. How can we run tests in parallel in TestNG?**

* **Answer:** You can configure TestNG to run tests in parallel by modifying the testng.xml file. You can specify the parallel execution at different levels (suite, test, or class).

<suite name="Suite" parallel="tests" thread-count="2">

<test name="Test1">

<classes>

<class name="TestClass"/>

</classes>

</test>

</suite>

**7. What is the difference between @BeforeMethod and @BeforeClass annotations?**

* **Answer:**
  + @BeforeMethod: Runs before each test method in the class.
  + @BeforeClass: Runs only once before the first test method in the class.

**8. What is a dependency in TestNG?**

* **Answer:** TestNG allows you to specify dependencies between test methods. This is done using the dependsOnMethods attribute. It ensures that a test method runs only if the dependent test methods have passed.

@Test

public void test1() {

// some test code

}

@Test(dependsOnMethods = "test1")

public void test2() {

// this will run only if test1 passes

}

**9. How would you handle exceptions in TestNG?**

* **Answer:** TestNG provides the expectedExceptions attribute in the @Test annotation to check if a specific exception is thrown. If the exception is thrown during the test execution, it will be considered as a successful test.

@Test(expectedExceptions = ArithmeticException.class)

public void testException() {

int result = 1 / 0; // this will throw an ArithmeticException

}

**10. What is the use of @Factory annotation in TestNG?**

* **Answer:** The @Factory annotation is used to create multiple instances of a test class. A factory method returns an array of objects of the test class, allowing you to run the same test with different test data.

@Factory

public Object[] createTestInstances() {

return new Object[] { new TestClass("Test 1"), new TestClass("Test 2") };

}

**11. What is the difference between @Test(enabled = false) and @Test(invocationCount = 0)?**

* **Answer:**
  + @Test(enabled = false): Disables the test method, and it will not be executed.
  + @Test(invocationCount = 0): This will not run the test method at all, as the invocation count is set to 0.

**12. What is the @Listeners annotation in TestNG?**

* **Answer:** The @Listeners annotation is used to specify listener classes in TestNG. Listeners allow you to track the progress of tests, such as when they start or fail, and can be used to implement logging, reporting, etc.

@Listeners(MyListener.class)

public class MyTestClass {

// Test methods

}

**13. What is the use of testng.xml file?**

* **Answer:** The testng.xml file is used to configure and organize test execution. It allows you to specify which tests or classes to run, set priorities, group tests, and configure parallel execution, among other things.

**14. Can TestNG be used for both unit testing and integration testing?**

* **Answer:** Yes, TestNG can be used for both unit testing and integration testing. It provides features like dependency testing, parallel execution, and supports annotations for setting up different stages of test execution, which makes it versatile for various types of testing.

**15. How do you skip a test in TestNG?**

* **Answer:** You can skip a test using @Test(enabled = false) or by throwing a SkipException within the test method.

@Test

public void testToSkip() {

throw new SkipException("Skipping this test");

}

**16. How would you handle test failures in TestNG?**

* **Answer:** In TestNG, you can handle test failures using the @AfterMethod annotation to perform some actions after every test method execution. You can also use listeners such as ITestListener to capture the test result and handle failures programmatically.

@AfterMethod

public void afterMethod(ITestResult result) {

if (result.getStatus() == ITestResult.FAILURE) {

// Handle failure

System.out.println("Test failed: " + result.getName());

}

}

**17. Explain TestNG’s @BeforeSuite and @AfterSuite annotations.**

* **Answer:**
  + @BeforeSuite: This annotation is used to run a method before the execution of any test in the test suite (defined in the XML file).
  + @AfterSuite: This annotation is used to run a method after the completion of all tests in the test suite.

@BeforeSuite

public void beforeSuite() {

System.out.println("Before Suite execution");

}

@AfterSuite

public void afterSuite() {

System.out.println("After Suite execution");

}

**18. How do you run a specific test method from the testng.xml file?**

* **Answer:** You can specify the test method to run by adding a <method> element under the <test> section in the testng.xml file.

<suite name="Suite">

<test name="Test1">

<classes>

<class name="TestClass">

<methods>

<include name="testMethod"/>

</methods>

</class>

</classes>

</test>

</suite>

**19. What is the invocationCount attribute in TestNG, and how is it used?**

* **Answer:** The invocationCount attribute in the @Test annotation is used to specify how many times a test method should be executed. This is useful for load testing or running the same test multiple times with different inputs or to check for flaky tests.

@Test(invocationCount = 5)

public void testMethod() {

System.out.println("Running test method");

}

**20. What is the timeOut attribute in TestNG, and how is it used?**

* **Answer:** The timeOut attribute in the @Test annotation specifies the maximum time (in milliseconds) that a test method is allowed to run. If the test exceeds this time, it will be considered a failure.

@Test(timeOut = 1000) // 1 second timeout

public void testMethod() throws InterruptedException {

Thread.sleep(2000); // This will cause the test to fail due to timeout

}

**21. What are the different ways to parameterize tests in TestNG?**

* **Answer:** TestNG allows parameterizing tests in two main ways:
  1. **Using @DataProvider**: Provides multiple sets of data to a test method.
  2. **Using @Parameters with testng.xml**: Parameters are passed from the XML configuration file to the test methods.

Example using @Parameters in testng.xml:

<suite name="Suite">

<test name="Test">

<parameter name="username" value="testuser"/>

<classes>

<class name="TestClass"/>

</classes>

</test>

</suite>

And in the test class:

@Test

@Parameters("username")

public void testMethod(String username) {

System.out.println(username); // Output: testuser

}

**22. What is the role of @Test(priority) in TestNG?**

* **Answer:** The @Test(priority) attribute in TestNG allows you to define the order in which test methods are executed. TestNG will run methods with a lower priority number first.

@Test(priority = 1)

public void testFirst() {

System.out.println("First test");

}

@Test(priority = 2)

public void testSecond() {

System.out.println("Second test");

}

**23. What is the @BeforeGroups and @AfterGroups annotation in TestNG?**

* **Answer:** These annotations are used to execute setup and teardown logic before and after a group of tests, respectively. You define groups in the testng.xml file, and the methods annotated with @BeforeGroups and @AfterGroups will be executed based on those groups.

@BeforeGroups("group1")

public void beforeGroup() {

System.out.println("Before group1 tests");

}

@AfterGroups("group1")

public void afterGroup() {

System.out.println("After group1 tests");

}

**24. What is the @Test(dependsOnGroups) attribute in TestNG?**

* **Answer:** The @Test(dependsOnGroups) attribute is used to specify that a test method depends on the successful execution of a group of tests. The test will only run if all the tests in the specified group have passed.

@Test(groups = "group1")

public void testMethod1() {

// Some logic

}

@Test(dependsOnGroups = "group1")

public void testMethod2() {

// This test will run only if all tests in "group1" pass

}

**25. How does TestNG handle test case retries?**

* **Answer:** TestNG provides the IRetryAnalyzer interface to implement custom retry logic for failed tests. You can use this interface to retry a failed test method a specified number of times before it is marked as failed.

public class RetryAnalyzer implements IRetryAnalyzer {

private int retryCount = 0;

private static final int maxRetryCount = 3;

@Override

public boolean retry(ITestResult result) {

if (retryCount < maxRetryCount) {

retryCount++;

return true;

}

return false;

}

}

@Test(retryAnalyzer = RetryAnalyzer.class)

public void testMethod() {

// This test will be retried 3 times in case of failure

}

**26. What is a TestNG listener and how do you use it?**

* **Answer:** A TestNG listener is an interface that allows you to hook into various test execution events (like test start, success, failure, etc.). There are multiple types of listeners available, such as ITestListener, ISuiteListener, IInvokedMethodListener, etc. You implement these interfaces to capture test execution details and take appropriate actions like logging, reporting, or sending notifications.

Example:

@Listeners(MyTestListener.class)

public class MyTestClass {

// Test methods

}

Custom Listener:

public class MyTestListener implements ITestListener {

@Override

public void onTestSuccess(ITestResult result) {

System.out.println("Test passed: " + result.getName());

}

@Override

public void onTestFailure(ITestResult result) {

System.out.println("Test failed: " + result.getName());

}

}

**27. What is the difference between @BeforeMethod and @BeforeTest?**

* **Answer:**
  + @BeforeMethod: Runs before each individual test method within a class.
  + @BeforeTest: Runs before the first test method of the specified test in testng.xml, regardless of the number of methods in the test.

**28. What are the ways to report test results in TestNG?**

* **Answer:**
  + **XML Reports**: TestNG provides a default XML report for each test run.
  + **HTML Reports**: TestNG generates detailed HTML reports with information about each test method.
  + **Custom Listeners**: You can use ITestListener or ISuiteListener to generate custom reports.

**Cucumber interview questions with answers**:

1. **What is Cucumber, and how does it work?**
   * Cucumber is a **Behavior-Driven Development (BDD)** tool that allows writing test cases in plain English using **Gherkin syntax**. It maps human-readable test scenarios to automation code using step definitions.
2. **Explain Gherkin syntax in Cucumber.**
   * Gherkin is a structured format for writing test cases in Cucumber. It uses keywords like:
   * Feature: Login functionality
   * Scenario: Valid login
   * Given User is on login page
   * When User enters valid credentials
   * Then User should be redirected to the dashboard
3. **What are feature files in Cucumber?**
   * A feature file contains **scenarios** written in Gherkin syntax. It describes application behavior in a structured way.
4. **What are step definitions in Cucumber?**
   * Step definitions link Gherkin steps to Java/Selenium code. Example:
   * @Given("User is on login page")
   * public void user\_is\_on\_login\_page() {
   * driver.get("https://example.com/login");
   * }
5. **What is the purpose of tags in Cucumber?**
   * Tags (@smoke, @regression) allow executing specific test scenarios selectively using the cucumber.options parameter. Example:
   * @smoke
   * Scenario: Valid login

**Intermediate Questions**

1. **How do you handle parameterization in Cucumber?**
   * Using **Examples table** in Scenario Outline or **Cucumber Expressions** in step definitions.
   * Example with Scenario Outline:
   * Scenario Outline: Login with multiple users
   * Given User is on login page
   * When User enters "<username>" and "<password>"
   * Then User should see the dashboard
   * Examples:
   * | username | password |
   * | user1 | pass123 |
   * | user2 | pass456 |
   * Step Definition:
   * @When("User enters {string} and {string}")
   * public void user\_enters\_credentials(String username, String password) {
   * loginPage.enterCredentials(username, password);
   * }
2. **What is a background in Cucumber, and when do you use it?**
   * Background is used for **common preconditions** shared by multiple scenarios.
   * Example:
   * Feature: User Login
   * Background:
   * Given User is on login page
   * Scenario: Valid login
   * When User enters valid credentials
   * Then User should see the dashboard
3. **How do you execute only specific scenarios in Cucumber?**
   * Using **tags** in the command line or cucumber.options:
   * mvn test -Dcucumber.options="--tags @smoke"
4. **How do you integrate Cucumber with TestNG/JUnit?**
   * Extend the Cucumber class and annotate it with @CucumberOptions:
   * @RunWith(Cucumber.class)
   * @CucumberOptions(features="src/test/resources/features", glue="stepDefinitions")
   * public class TestRunner { }
5. **Explain hooks in Cucumber (Before, After).**
   * Hooks (@Before, @After) allow setting up and tearing down tests.
   * Example:
   * @Before
   * public void setup() { driver = new ChromeDriver(); }
   * @After
   * public void teardown() { driver.quit(); }

**Advanced Questions**

1. **How do you handle data-driven testing in Cucumber?**
   * Using **Scenario Outline + Examples table** or reading data from **Excel/CSV** using Apache POI.
2. **What is Scenario Outline in Cucumber?**
   * It allows running a scenario multiple times with different data sets.
   * Example:
   * Scenario Outline: Login with multiple users
   * Given User is on login page
   * When User enters "<username>" and "<password>"
   * Then User should see the dashboard
   * Examples:
   * | username | password |
   * | user1 | pass123 |
   * | user2 | pass456 |
3. **How do you integrate Cucumber with Selenium?**
   * Use Selenium WebDriver inside step definitions. Example:
   * @Given("User is on login page")
   * public void user\_is\_on\_login\_page() {
   * driver.get("https://example.com/login");
   * }
4. **How do you generate reports in Cucumber?**
   * Use cucumber-reporting or ExtentReports. Example:
   * @CucumberOptions(
   * plugin = {"pretty", "html:target/cucumber-reports"}
   * )
5. **How do you handle API testing with Cucumber and Rest Assured?**
   * Use Rest Assured inside step definitions. Example:
   * @When("User sends GET request to {string}")
   * public void user\_sends\_get\_request(String url) {
   * response = given().when().get(url);
   * }

**API testing interview questions with answers**:

1. **What is API testing?**
   * API testing involves testing the **request and response** of application interfaces to ensure data accuracy, security, performance, and reliability.
2. **What are common HTTP methods used in API testing?**
   * **GET** – Retrieve data
   * **POST** – Create data
   * **PUT** – Update data
   * **DELETE** – Remove data
   * **PATCH** – Partially update data
3. **What is the difference between SOAP and REST APIs?**
   * **SOAP**: XML-based, strict standards, uses WSDL.
   * **REST**: JSON/XML, flexible, stateless, lightweight.
4. **What are the components of an HTTP request?**
   * **Method (GET, POST, etc.)**
   * **URL**
   * **Headers** (Authorization, Content-Type)
   * **Body** (for POST, PUT requests)
5. **What is the difference between authentication and authorization?**
   * **Authentication**: Verifies identity (e.g., login).
   * **Authorization**: Grants access to resources (e.g., roles, permissions).

**Intermediate Questions**

1. **How do you test an API using Postman?**
   * Set method (GET, POST, etc.), provide URL, add headers, body (if needed), send request, and validate the response.
2. **What is status code 200, 201, 400, 401, 403, 500?**
   * **200** – OK
   * **201** – Created
   * **400** – Bad Request
   * **401** – Unauthorized
   * **403** – Forbidden
   * **500** – Internal Server Error
3. **How do you validate API response?**
   * Check **status code, response body, headers, response time**.
4. **What is JSON Schema validation in API testing?**
   * Ensures response structure matches expected format. Example using Postman:
   * {
   * "name": "John",
   * "age": 30
   * }

Schema:

{

"type": "object",

"properties": {

"name": {"type": "string"},

"age": {"type": "integer"}

}

}

1. **How do you handle authentication in API testing?**
   * **Basic Auth** (Username/Password)
   * **Bearer Token** (OAuth, JWT)
   * **API Key**
   * **OAuth 2.0**

**Advanced Questions**

1. **How do you automate API testing using Rest Assured?**
2. given()
3. .header("Authorization", "Bearer token")
4. .when()
5. .get("https://api.example.com/users")
6. .then()
7. .statusCode(200);
8. **How do you perform parameterization in API testing?**
   * Using **Query Parameters** (?id=123) or **Path Parameters** (/users/{id}).
9. given().pathParam("id", 123)
10. .when().get("/users/{id}")
11. .then().statusCode(200);
12. **What is API rate limiting?**
    * Restricts the number of API requests per user/IP to prevent overloading (e.g., **1000 requests per hour**).
13. **What are idempotent methods in REST APIs?**
    * **GET, PUT, DELETE** are idempotent (same request gives same response).
    * **POST** is **not** idempotent (creates new records each time).
14. **What are common API security vulnerabilities?**
    * **SQL Injection**
    * **Cross-Site Scripting (XSS)**
    * **Broken Authentication**
    * **Exposed Sensitive Data**

**Database Testing Interview Questions & Answers**

**Basic Questions**

1. **What is database testing?**
   * Database testing ensures data integrity, consistency, and performance by verifying **schemas, tables, triggers, stored procedures, and transactions**.
2. **What are the types of database testing?**
   * **Functional Testing** – Validates data storage & retrieval
   * **Performance Testing** – Checks query execution time
   * **Security Testing** – Ensures access control
   * **Data Integrity Testing** – Verifies correctness across tables
3. **What SQL commands are used in database testing?**
   * **DDL (Data Definition Language)** – CREATE, ALTER, DROP
   * **DML (Data Manipulation Language)** – INSERT, UPDATE, DELETE, SELECT
   * **DCL (Data Control Language)** – GRANT, REVOKE
   * **TCL (Transaction Control Language)** – COMMIT, ROLLBACK, SAVEPOINT
4. **What are primary and foreign keys?**
   * **Primary Key**: Uniquely identifies a record (e.g., id in users table).
   * **Foreign Key**: Links two tables (e.g., user\_id in orders table).
5. **How do you check if two tables have the same data?**
6. SELECT \* FROM table1
7. EXCEPT
8. SELECT \* FROM table2;
   * If the result is empty, both tables have the same data.

**Intermediate Questions**

1. **How do you validate data integrity in database testing?**
   * **Check constraints** (Primary Key, Foreign Key, Unique, Not Null)
   * **Use Joins** to verify relationships
   * **Compare data before & after transactions**
2. **How do you test stored procedures?**
   * Pass **input parameters**, check **output values**, and validate **data changes** using SQL.
3. EXEC procedure\_name @param1 = 'value1';
4. **What is indexing in databases?**
   * Indexing improves query performance by storing sorted data for faster retrieval.
5. CREATE INDEX idx\_name ON users (email);
6. **How do you check for duplicate records?**
7. SELECT column\_name, COUNT(\*)
8. FROM table\_name
9. GROUP BY column\_name
10. HAVING COUNT(\*) > 1;
11. **What is normalization, and why is it important?**

* **Normalization** removes redundancy & improves efficiency.
* **1NF**: No duplicate columns, atomic values.
* **2NF**: No partial dependencies.
* **3NF**: No transitive dependencies.

**Advanced Questions**

1. **How do you perform database performance testing?**

* Use **EXPLAIN PLAN** to analyze query execution.

EXPLAIN ANALYZE SELECT \* FROM orders WHERE user\_id = 123;

* Use indexing & optimize queries.

1. **What is ACID in database transactions?**

* **Atomicity** – Transaction is all or nothing.
* **Consistency** – Data must remain valid.
* **Isolation** – Transactions don’t affect each other.
* **Durability** – Data is permanently stored.

1. **How do you test data migration from one database to another?**

* **Row count comparison**
* **Data type validation**
* **Referential integrity check**
* **Using MINUS operator in SQL**

SELECT \* FROM source\_table

MINUS

SELECT \* FROM target\_table;

1. **How do you handle NULL values in database testing?**

* Use **IS NULL** or **COALESCE()** to handle NULLs.

SELECT COALESCE(column\_name, 'Default Value') FROM table\_name;

1. **How do you automate database testing?**

* Use tools like **Selenium with JDBC, Apache JMeter, DbUnit, SQL scripts in TestNG/JUnit**.

Connection con = DriverManager.getConnection(url, user, password);

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM users");