**Module 3 (Testing on Live Application)**

**What is RDBMS**

RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd in 1970.

**What is SQL**

**SQL** is a database computer language designed for the retrieval and management of data in a relational database like MySQL, MS Access, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres etc. **SQL** stands for **Structured Query Language**. SQL was developed in the 1970s by IBM Computer Scientists.

**SQL** is not a database management system, but it is a query language which is used to store and retrieve the data from a database or in simple words SQL is a language that communicates with databases.

Applications of SQL

SQL is one of the most widely used Query Language over the databases. SQL provides following functionality to the database programmers:

* Execute different database queries against a database.
* Define the data in a database and manipulate that data.
* Create data in a relational database management system.
* Access data from the relational database management system.
* Create and drop databases and tables.
* Create and maintain database users.
* Create view, stored procedure, functions in a database.
* Set permissions on tables, procedures and views.

**Write SQL Commands**

* SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
* SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

## Types of SQL Commands

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



### **1. Data Definition Language (DDL)**

* DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
* All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

* CREATE
* ALTER
* DROP
* TRUNCATE

**a. CREATE** It is used to create a new table in the database.

**Syntax:**

1. CREATE TABLE TABLE\_NAME (COLUMN\_NAME DATATYPES[,....]);

**Example:**

1. CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

**b. DROP:** It is used to delete both the structure and record stored in the table.

**Syntax**

1. DROP TABLE table\_name;

**Example**

1. DROP TABLE EMPLOYEE;

**c. ALTER:** It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

**Syntax:**

To add a new column in the table

1. ALTER TABLE table\_name ADD column\_name COLUMN-definition;

To modify existing column in the table:

1. ALTER TABLE table\_name MODIFY(column\_definitions....);

**EXAMPLE**

1. ALTER TABLE STU\_DETAILS ADD(ADDRESS VARCHAR2(20));
2. ALTER TABLE STU\_DETAILS MODIFY (NAME VARCHAR2(20));

**d. TRUNCATE:** It is used to delete all the rows from the table and free the space containing the table.

**Syntax:**

1. TRUNCATE TABLE table\_name;

**Example:**

1. TRUNCATE TABLE EMPLOYEE;

### **2. Data Manipulation Language**

* DML commands are used to modify the database. It is responsible for all form of changes in the database.
* The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

* INSERT
* UPDATE
* DELETE

**a. INSERT:** The INSERT statement is a SQL query. It is used to insert data into the row of a table.

**Syntax:**

1. INSERT INTO TABLE\_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, .... valueN);

Or

1. INSERT INTO TABLE\_NAME
2. VALUES (value1, value2, value3, .... valueN);

**For example:**

1. INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");

**b. UPDATE:** This command is used to update or modify the value of a column in the table.

**Syntax:**

1. UPDATE table\_name SET [column\_name1= value1,...column\_nameN = valueN] [WHERE CONDITION]

**For example:**

1. UPDATE students
2. SET User\_Name = 'Sonoo'
3. WHERE Student\_Id = '3'

**c. DELETE:** It is used to remove one or more row from a table.

**Syntax:**

1. DELETE FROM table\_name [WHERE condition];

**For example:**

1. DELETE FROM javatpoint
2. WHERE Author="Sonoo";

### **3. Data Control Language**

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

* Grant
* Revoke

**a. Grant:** It is used to give user access privileges to a database.

**Example**

1. GRANT SELECT, UPDATE ON MY\_TABLE TO SOME\_USER, ANOTHER\_USER;

**b. Revoke:** It is used to take back permissions from the user.

**Example**

1. REVOKE SELECT, UPDATE ON MY\_TABLE FROM USER1, USER2;

### **4. Transaction Control Language**

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

* COMMIT
* ROLLBACK
* SAVEPOINT

**a. Commit:** Commit command is used to save all the transactions to the database.

**Syntax:**

1. COMMIT;

**Example:**

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;

**b. Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.

**Syntax:**

1. ROLLBACK;

**Example:**

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;

**c. SAVEPOINT:** It is used to roll the transaction back to a certain point without rolling back the entire transaction.

**Syntax:**

1. SAVEPOINT SAVEPOINT\_NAME;

### **5. Data Query Language**

DQL is used to fetch the data from the database.

It uses only one command:

* SELECT

**a. SELECT:** This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

**Syntax:**

1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;

**For example:**

1. SELECT emp\_name
2. FROM employee
3. WHERE age > 20;

**What is join?**

In real life, we store our data in multiple logical tables that are linked together by a common key value in relational databases like SQL Server, [Oracle](https://www.javatpoint.com/oracle-tutorial), [MySQL](https://www.javatpoint.com/mysql-tutorial), and others. As a result, we constantly need to get data from two or more tables into the desired output based on some conditions. We can quickly achieve this type of data in SQL Server using the SQL JOIN clause. This article gives a complete overview of JOIN and its different types with an example.

The join clause allows us to **retrieve data from two or more related tables** into a meaningful result set. We can join the table using a **SELECT** statement and a **join condition**. It indicates how SQL Server can use data from one table to select rows from another table. In general, tables are related to each other using **foreign key** constraints.

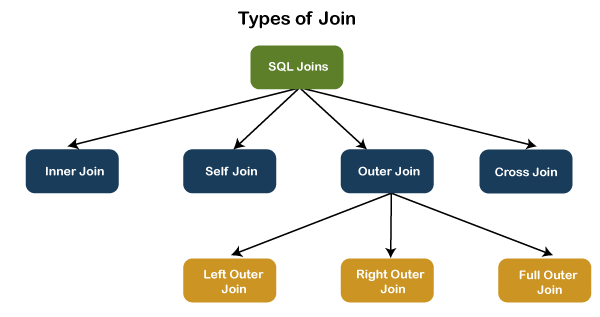
In a JOIN query, a condition indicates how two tables are related:

* Choose columns from each table that should be used in the join. A join condition indicates a foreign key from one table and its corresponding key in the other table.
* Specify the logical operator to compare values from the columns like =, <, or >.

**Write type of joins.**

[SQL Server](https://www.javatpoint.com/sql-server-tutorial) mainly supports **four types of JOINS**, and each join type defines how two tables are related in a query. The following are types of join supports in SQL Server:

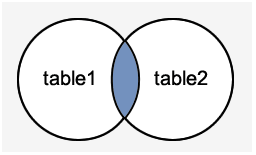
1. INNER JOIN
2. SELF JOIN
3. CROSS JOIN
4. OUTER JOIN



## INNER JOIN

This JOIN returns all records from multiple tables that satisfy the specified join condition. It is the simple and most popular form of join and assumes as a **default join**. If we omit the INNER keyword with the JOIN query, we will get the same output.

The following visual representation explains how INNER JOIN returns the matching records from **table1** and **table2:**



**INNER JOIN Syntax**

The following syntax illustrates the use of INNER JOIN in SQL Server:

1. **SELECT** columns
2. **FROM** table1
3. **INNER** JOIN table2 **ON** condition1
4. **INNER** JOIN table3 **ON** condition2

**INNER JOIN Example**

Let us first create two tables "**Student**" and "**Fee**" using the following statement:

1. **CREATE** **TABLE** Student (
2. id **int** **PRIMARY** **KEY** IDENTITY,
3. admission\_no **varchar**(45) NOT NULL,
4. first\_name **varchar**(45) NOT NULL,
5. last\_name **varchar**(45) NOT NULL,
6. age **int**,
7. city **varchar**(25) NOT NULL
8. );
10. **CREATE** **TABLE** Fee (
11. admission\_no **varchar**(45) NOT NULL,
12. course **varchar**(45) NOT NULL,
13. amount\_paid **int**,
14. );

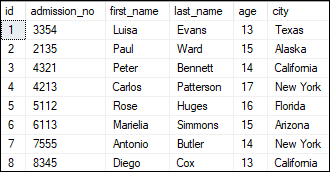
Next, we will insert some records into these tables using the below statements:

1. **INSERT** **INTO** Student (admission\_no, first\_name, last\_name, age, city)
2. **VALUES** (3354,'Luisa', 'Evans', 13, 'Texas'),
3. (2135, 'Paul', 'Ward', 15, 'Alaska'),
4. (4321, 'Peter', 'Bennett', 14, 'California'),
5. (4213,'Carlos', 'Patterson', 17, 'New York'),
6. (5112, 'Rose', 'Huges', 16, 'Florida'),
7. (6113, 'Marielia', 'Simmons', 15, 'Arizona'),
8. (7555,'Antonio', 'Butler', 14, 'New York'),
9. (8345, 'Diego', 'Cox', 13, 'California');

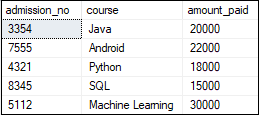
12. **INSERT** **INTO** Fee (admission\_no, course, amount\_paid)
13. **VALUES** (3354,'Java', 20000),
14. (7555, 'Android', 22000),
15. (4321, 'Python', 18000),
16. (8345,'SQL', 15000),
17. (5112, 'Machine Learning', 30000);

Execute the **SELECT** statement to verify the records:

**Table: Student**



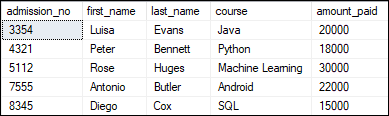
**Table: Fee**



We can demonstrate the INNER JOIN using the following command:

1. **SELECT** Student.admission\_no, Student.first\_name, Student.last\_name, Fee.course, Fee.amount\_paid
2. **FROM** Student
3. **INNER** JOIN Fee
4. **ON** Student.admission\_no = Fee.admission\_no;

This command gives the below result:



In this example, we have used the **admission\_no column** as a join condition to get the data from both tables. Depending on this table, we can see the information of the students who have paid their fee.

## SELF JOIN

A table is joined to itself using the SELF JOIN. It means that **each table row is combined with itself** and with every other table row. The SELF JOIN can be thought of as a JOIN of two copies of the same tables. We can do this with the help of **table name aliases** to assign a specific name to each table's instance. The table aliases enable us to use the **table's temporary** name that we are going to use in the query. It's a useful way to extract hierarchical data and comparing rows inside a single table.

**SELF JOIN Syntax**

The following expression illustrates the syntax of SELF JOIN in SQL Server. It works the same as the syntax of joining two different tables. Here, we use aliases names for tables because both the table name are the same.

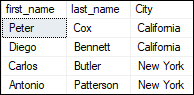
1. **SELECT** T1.col\_name, T2.col\_name...
2. **FROM** table1 T1, table1 T2
3. **WHERE** join\_condition;

**Example**

We can demonstrate the SELF JOIN using the following command:

1. **SELECT** S1.first\_name, S2.last\_name, S2.city
2. **FROM** Student S1, Student S2
3. **WHERE** S1.id <> S2.iD AND S1.city = S2.city
4. **ORDER** **BY** S2.city;

This command gives the below result:

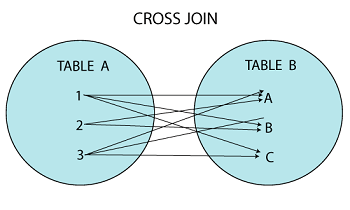


In this example, we have used the **id and city column** as a join condition to get the data from both tables.

## CROSS JOIN

CROSS JOIN in SQL Server combines all of the possibilities of two or more tables and returns a result that includes every row from all contributing tables. It's also known as **CARTESIAN JOIN** because it produces the **Cartesian product** of all linked tables. The Cartesian product represents all rows present in the first table multiplied by all rows present in the second table.

The below visual representation illustrates the CROSS JOIN. It will give all the records from **table1** and **table2** where each row is the combination of rows of both tables:



**CROSS JOIN Syntax**

The following syntax illustrates the use of CROSS JOIN in SQL Server:

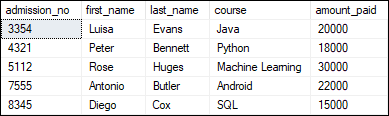
1. **SELECT** column\_lists
2. **FROM** table1
3. CROSS JOIN table2;

**Example**

We can demonstrate the CROSS JOIN using the following command:

1. **SELECT** Student.admission\_no, Student.first\_name, Student.last\_name, Fee.course, Fee.amount\_paid
2. **FROM** Student
3. CROSS JOIN Fee
4. **WHERE** Student.admission\_no = Fee.admission\_no;

This command gives the below result:



## OUTER JOIN

OUTER JOIN in SQL Server **returns all records from both tables** that satisfy the join condition. In other words, this join will not return only the matching record but also return all unmatched rows from one or both tables.

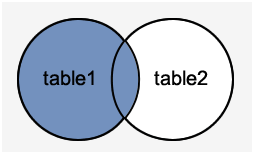
**We can categories the OUTER JOIN further into three types:**

* LEFT OUTER JOIN
* RIGHT OUTER JOIN
* FULL OUTER JOIN

### **LEFT OUTER JOIN**

The LEFT OUTER JOIN **retrieves all the records from the left table and matching rows from the right table**. It will return **NULL** when no matching record is found in the right side table. Since OUTER is an optional keyword, it is also known as LEFT JOIN.

The below visual representation illustrates the LEFT OUTER JOIN:



**LEFT OUTER JOIN Syntax**

The following syntax illustrates the use of LEFT OUTER JOIN in SQL Server:

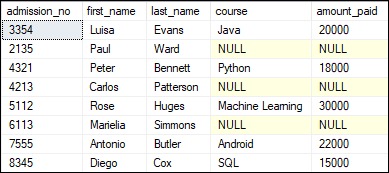
1. **SELECT** column\_lists
2. **FROM** table1
3. LEFT [OUTER] JOIN table2
4. **ON** table1.**column** = table2.**column**;

**Example**

We can demonstrate the LEFT OUTER JOIN using the following command:

1. **SELECT** Student.admission\_no, Student.first\_name, Student.last\_name, Fee.course, Fee.amount\_paid
2. **FROM** Student
3. LEFT OUTER JOIN Fee
4. **ON** Student.admission\_no = Fee.admission\_no;

This command gives the below result:

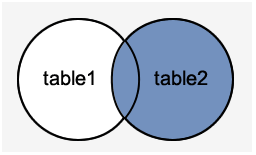


This output shows that the unmatched row's values are replaced with NULLs in the respective columns.

### **RIGHT OUTER JOIN**

The RIGHT OUTER JOIN **retrieves all the records from the right-hand table and matched rows from the left-hand table**. It will return **NULL** when no matching record is found in the left-hand table. Since OUTER is an optional keyword, it is also known as RIGHT JOIN.

The below visual representation illustrates the RIGHT OUTER JOIN:



**RIGHT OUTER JOIN Syntax**

The following syntax illustrates the use of RIGHT OUTER JOIN in SQL Server:

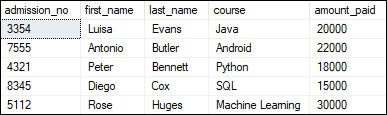
1. **SELECT** column\_lists
2. **FROM** table1
3. RIGHT [OUTER] JOIN table2
4. **ON** table1.**column** = table2.**column**;

**Example**

The following example explains how to use the RIGHT OUTER JOIN to get records from both tables:

1. **SELECT** Student.admission\_no, Student.first\_name, Student.last\_name, Fee.course, Fee.amount\_paid
2. **FROM** Student
3. RIGHT OUTER JOIN Fee
4. **ON** Student.admission\_no = Fee.admission\_no;

This command gives the below result:

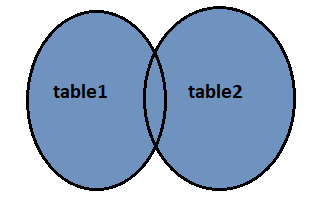


In this output, we can see that no column has NULL values because all rows in the Fee table are available in the Student table based on the specified condition.

### **FULL OUTER JOIN**

The FULL OUTER JOIN in SQL Server **returns a result that includes all rows from both tables**. The columns of the right-hand table return NULL when no matching records are found in the left-hand table. And if no matching records are found in the right-hand table, the left-hand table column returns NULL.

The below visual representation illustrates the FULL OUTER JOIN:



**FULL OUTER JOIN Syntax**

The following syntax illustrates the use of FULL OUTER JOIN in SQL Server:

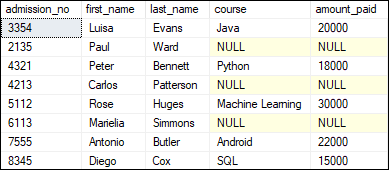
1. **SELECT** column\_lists
2. **FROM** table1
3. **FULL** [OUTER] JOIN table2
4. **ON** table1.**column** = table2.**column**;

**Example**

The following example explains how to use the FULL OUTER JOIN to get records from both tables:

1. **SELECT** Student.admission\_no, Student.first\_name, Student.last\_name, Fee.course, Fee.amount\_paid
2. **FROM** Student
3. **FULL** OUTER JOIN Fee
4. **ON** Student.admission\_no = Fee.admission\_no;

This command gives the below result:



In this output, we can see that the column has NULL values when no matching records are found in the left-hand and right-hand table based on the specified condition.

**How Many constraint and describes itself**

Constraints in SQL means we are applying certain conditions or restrictions on the database. This further means that before inserting data into the database, we are checking for some conditions. If the condition we have applied to the database holds true for the data which is to be inserted, then only the data will be inserted into the database tables.

### **Constraints in SQL can be categorized into two types:**

1. **Column Level Constraint:**  
   Column Level Constraint is used to apply a constraint on a single column.
2. **Table Level Constraint:**  
   Table Level Constraint is used to apply a constraint on multiple columns.

### **Some of the real-life examples of constraints are as follows:**

1. Every person has a unique email id. This is because while creating an email account for any user, the email providing services such as Gmail, Yahoo or any other email providing service will always check for the availability of the email id that the user wants for himself. If some other user already takes the email id that the user wants, then that id cannot be assigned to another user. This simply means that no two users can have the same email ids on the same email providing service. So, here the email id is the constraint on the database of email providing services.
2. Whenever we set a password for any system, there are certain constraints that are to be followed. These constraints may include the following:
   * There must be one uppercase character in the password.
   * Password must be of at least eight characters in length.
   * Password must contain at least one special symbol.

### **Constraints available in SQL are:**

1. NOT NULL
2. UNIQUE
3. PRIMARY KEY
4. FOREIGN KEY
5. CHECK
6. DEFAULT
7. CREATE INDEX

**Difference between RDBMS vs DBMS**

**Database Management System (DBMS)** is a software that is used to define, create and maintain a database and provides controlled access to the data.

**Relational Database Management System (RDBMS)** is an advanced version of a DBMS. 

| **DBMS** | **RDBMS** |
| --- | --- |
| [DBMS](https://www.geeksforgeeks.org/introduction-of-dbms-database-management-system-set-1/) stores data as file. | [RDBMS](https://www.geeksforgeeks.org/rdbms-architecture/) stores data in tabular form. |
| Data elements need to access individually. | Multiple data elements can be accessed at the same time. |
| No relationship between data. | Data is stored in the form of tables which are related to each other. |
| Normalization is not present. | Normalization is present. |
| DBMS does not support distributed database. | RDBMS supports distributed database. |
| It stores data in either a navigational or hierarchical form. | It uses a tabular structure where the headers are the column names, and the rows contain corresponding values. |
| It deals with small quantity of data. | It deals with large amount of data. |
| Data redundancy is common in this model. | Keys and indexes do not allow Data redundancy. |
| It is used for small organization and deal with small data. | It is used to handle large amount of data. |
| It supports single user. | It supports multiple users. |
| Data fetching is slower for the large amount of data. | Data fetching is fast because of relational approach. |
| The data in a DBMS is subject to low security levels with regards to data manipulation. | There exists multiple levels of data security in a RDBMS. |
| Low software and hardware necessities. | Higher software and hardware necessities. |
| Examples:[XML](https://www.geeksforgeeks.org/xml-basics/), Window Registry, etc. | Examples: [MySQL](https://www.geeksforgeeks.org/architecture-of-mysql/), [PostgreSQL](https://www.geeksforgeeks.org/what-is-postgresql-introduction/), SQL Server, Oracle, Microsoft Access etc. |

**What is API Testing**

API stands for **A**pplication **P**rogramming **I**nterface, which specifies how one component should interact with the other. It consists of a set of routines, protocols and tools for building the software applications.

The API Testing is performed for the system, which has a collection of API that ought to be tested. During Testing, a test of following things is looked at.

* Exploring boundary conditions and ensuring that the test harness varies parameters of the API calls in ways that verify functionality and expose failures.
* Generating more value added parameter combinations to verify the calls with two or more parameters.
* Verifying the behaviour of the API which is considering the external environment conditions such as files, peripheral devices, and so forth.
* Verifying the Sequence of API calls and check if the API's produce useful results from successive calls.

**Types of API Testing**

1. Functional Testing:

The very first API testing type on the list falls under the black-box testing category. This type of testing bases its test cases on the specifications of the software component under test; meaning that there is a test of specific functions within the codebase that stand for specific scenarios. These functions are designed for the sole purpose of monitoring whether the API is behaving as it is expected to.

They also help ensure that the deviations, if any, are dealt with in a timely and effective manner. In its truest sense, functional testing is a quality assurance process where functions are tested by feeding them input and examining the output.

### 2. UI Testing:

As the name clearly suggests, [UI testing](https://www.cybersuccess.biz/difference-between-api-and-unit-testing/) is a specific process of ensuring the smooth functioning of the user interface. Often used as an alternative for functional testing, UI testing has more to do with the user interface associated with the API rather than the API in terms of the codebase. The main objective of this test is to provide developers with a quick analysis of the usability, efficiency, and functionality of both the front-end and the back-end.

Despite the fact that it makes for a viable alternative for functional testing, it is highly recommended that UI testing should be used only for testing functions related to the user interface, and should be treated as a subset during functional testing.

### 3. Runtime & Error Detection:

Most testing types like Functional testing and Validation testing involve analysing the API behaviour in the given environments or scenarios. On the other hand, runtime and error detection testing only concern itself with monitoring the actual running of the API. It enables the developers to identify the potential defects that are detected during execution.

When implemented with utmost precision, the test can provide accurate results in terms of bug detection. These tests are your best bet when it comes to exposing critical bugs by focusing on specific aspects like monitoring, error detection, execution errors, and any possible leaks.

### 4. Load Testing:

This non-functional testing helps determine the performance of a software application or product. This test is carried out at the end of the completion of the codebase that brought the product into being. The software application is released in real-time environments where a number of users can test it at the same time.

To ensure enhanced effectiveness, load testing simulates different scenarios for testing the application and upgrading it to its highest potential. These tests help the developer understand the operating capability of the application, study the sustainability of the application when multiple users are using it, and determine the application’s capacity to scale in order to accommodate more users.

### 5. Security Testing:

One of the most critical testing types, Security Testing brings the weak links, loopholes, threats, and possible risks in a software application to the developers’ attention. The tests also help prevent harmful attacks from external threats. If not identified and rectified in time, these threats can lead to the leak of valuable information, revenue, and the very reputation of the organization.

When done correctly, security testing can combat the threats so that the system functions smoothly without external interference. After the detection of these vulnerabilities, developers can fix them and strengthen the application via coding.

### 6. Validation Testing:

Validation testing is one of the last processes that are carried out during the test cycle of a software application. It is conducted after verification of the API’s constituent parts and functions, at the end of the development process.

When it comes to effective validation testing, here are a few things developers must consider – has the product been built in accordance with the project specifications? Is the API an accurate and efficient method of doing what is required? Can any modifications be made to the existing codebase in order to optimize the performance of the software product? This set of basic questions is quite important to the successful execution of any and every application.

**What is Responsive Testing?**

Responsive testing involves how a website or web application looks and behaves on different devices, screen sizes, and resolutions. The goal of responsive testing is to ensure that the website or web application can be used effectively on various devices, including desktops, laptops, tablets, and smartphones.

**Which types of tools are available for Responsive Testing**

* Testsigma
* Responsinator
* Screenfly
* LambdaTest
* Am I Responsive?
* CrossBrowserTesting
* Browserstack

**What is the full form of. ipa, .apk**

An IPA (iOS App Store Package) file is an iOS application archive file that stores an iOS app. Each IPA file includes a binary and can only be installed on an iOS device.

APK file stands for (Android Application Package). APK is a file extension of an Android device. APK files can normally be used in Android and a number of other Android-based **Operating Systems** for the distribution and installation of mobile apps and mobile games.

**How to create step for to open the developer option mode ON?**

**Go to "Settings"**

**Tap "About device" or "About phone"**

**Tap “Software information”**

**Tap “Build number” seven times**

**Enter your pattern, PIN or password to enable the Developer options menu**

**The "Developer options" menu will now appear in your Settings menu**

To disable the Developer options menu, tap the switch