

PRACTICE

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

NOT NULL - Ensures that a column cannot have a NULL value

UNIQUE - Ensures that all values in a column are different

PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

FOREIGN KEY - Uniquely identifies a row/record in another table

CHECK - Ensures that all values in a column satisfies a specific condition

DEFAULT - Sets a default value for a column when no value is specified

INDEX - Used to create and retrieve data from the database very quickly

```
DataTable dt =  
user.GetBranchNameForAutoComplete(Configurations.Instance.ConnectionString,  
Convert.ToString(HttpContext.Current.Session["SapCode"]), SearchKey,  
Convert.ToString(HttpContext.Current.Session["Oprn_Category"]).Trim());  
  
dynamic ComMaster = (from DataRow m in dt.Rows  
select new  
{  
    BRANCH = Convert.ToString(m["f_Branch"]).Trim(),  
    BRNDES = Convert.ToString(m["f_BranchDesc"]).Trim()  
}).ToList();  
  
return JsonConvert.SerializeObject(new { Table = ComMaster, Code = 200 });
```

```
string num = "963587";
int Number = int.Parse(num);
int Reverse = 0;
while (Number > 0)
{
    int remainder = Number % 10;
    Reverse = (Reverse * 10) + remainder;
    Number = Number / 10;
}
int RevNo = Reverse;

string str = "", reverse = "";
int Length = 0;

//Getting String(word) from Console
str = "Pramit";
//Calculate length of string str
Length = str.Length - 1;
while (Length >= 0)
{
    reverse = reverse + str[Length];
    Length--;
}

string RevString = reverse;
```

```
select * from (
select * , ROW_NUMBER() over (order by f_Name) as RowNum from t_BVR_Students
) tvr where RowNum >1
```

Delete Duplicate Records

```
with CTE
as
(select ROW_NUMBER() over(partition by f_Name order by f_Name ) as RowNum, * from
t_BVR_Students )
delete from CTE where RowNum >1
```

Update Using Joins

```
update a
set a.f_Name =b.f_Name
from
t_BVR_Students a inner join t_BVR_Students_1 b on a.f_Name = b.f_Name
```

Fibonacci Series

```
1. using System;
2. public class FibonacciExample
3. {
4.     public static void Main(string[] args)
5.     {
6.         int n1=0,n2=1,n3,i,number;
7.         Console.Write("Enter the number of elements: ");
8.         number = int.Parse(Console.ReadLine());
9.         Console.Write(n1+" "+n2+" "); //printing 0 and 1
10.        for(i=2;i<number;++i) //loop starts from 2 because 0 and 1 are already printed
11.        {
12.            n3=n1+n2;
13.            Console.Write(n3+" ");
```

```

14.     n1=n2;
15.     n2=n3;
16.     }
17.     }
18.     }

```

Route Config

```

public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
        );
    }
}

```

WebApi Config

```

public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Web API configuration and services

        // Web API routes
        config.MapHttpAttributeRoutes();

        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}

```

Bundle Config

```

public class BundleConfig
{
    // For more information on bundling, visit
http://go.microsoft.com/fwlink/?LinkId=301862
    public static void RegisterBundles(BundleCollection bundles)

```

```

{
    bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
        "~/Scripts/jquery-{version}.js"));

    bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
        "~/Scripts/jquery.validate*"));

    // Use the development version of Modernizr to develop with and learn from.
    Then, when you're
    // ready for production, use the build tool at http://modernizr.com to pick
    only the tests you need.
    bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
        "~/Scripts/modernizr-*"));

    bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(
        "~/Scripts/bootstrap.js",
        "~/Scripts/respond.js"));

    bundles.Add(new StyleBundle("~/Content/css").Include(
        "~/Content/bootstrap.min.css",
        "~/Content/site.min.css"));
}
}

```

Filter Config

```

public class FilterConfig
{
    public static void RegisterGlobalFilters(GlobalFilterCollection filters)
    {
        filters.Add(new HandleErrorAttribute());
    }
}

```

Global.asax

```

protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();
    GlobalConfiguration.Configure(WebApiConfig.Register);
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
}

```

Calling in Master Page

```

@Styles.Render("~/Content/css")
@Scripts.Render("~/bundles/modernizr")

@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/bootstrap")
@RenderSection("scripts", required: false)

```

Calling in Page

```
@section scripts{
```

```
}
```

HTML.Begin Form

```
@model WebApplication1.Models.AddStudent
```

```
@{
```

```
    ViewBag.Title = "Create";
```

```
}
```

```
<h2>Create</h2>
```

```
@using (Html.BeginForm("Create", "Milnaa", "POST"))
```

```
{
```

```
    @Html.AntiForgeryToken()
```

```
    <div class="form-horizontal">
```

```
        <h4>AddStudent</h4>
```

```
        <hr />
```

```
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
```

```
        <div class="form-group">
```

```
            @Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-label col-md-2" })
```

```
            <div class="col-md-10">
```

```
                @Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class = "form-control" } })
```

```
                @Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-danger" })
```

```
            </div>
```

```
        </div>
```

```
        <div class="form-group">
```

```
            @Html.LabelFor(model => model.Email, htmlAttributes: new { @class = "control-label col-md-2" })
```

```
            <div class="col-md-10">
```

```
                @Html.EditorFor(model => model.Email, new { htmlAttributes = new { @class = "form-control" } })
```

```
                @Html.ValidationMessageFor(model => model.Email, "", new { @class = "text-danger" })
```

```
            </div>
```

```
        </div>
```

```
        <div class="form-group">
```

```
            @Html.LabelFor(model => model.Gender, htmlAttributes: new { @class = "control-label col-md-2" })
```

```
            <div class="col-md-10">
```

```
                @Html.EditorFor(model => model.Gender, new { htmlAttributes = new { @class = "form-control" } })
```

```
                @Html.ValidationMessageFor(model => model.Gender, "", new { @class = "text-danger" })
```

```
            </div>
```

```

        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
}

```

Controller Method With Validate Anti Forgery Token

```

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create(FormCollection form)
{
    return View();
}

```

The `$(document).ready()` is a jQuery event which occurs when the HTML document has been fully loaded, while the `window.onload` event occurs later, when everything including images on the page loaded. Also `window.onload` is a pure javascript event in the DOM, while the `$(document).ready()` event is a method in jQuery

```

$.ajax({
    url: '../shaft/GetData',
    type: 'POST',
    contentType: 'application/json',
    data: JSON.stringify(Input),
    success: function (result) {
        var data = JSON.parse(result);

        //if (data.count > 0)
        {
            $("#tbody_Append").html('');
            var _html = '';
            $.each(data, function (key, item) {
                _html += "<tr>";
                _html += "<td>" + item.ID + "</td>";
                _html += "<td id='Name_" + item.ID + "'>" + item.Name +
                _html += "<td id='Gender_" + item.ID + "'>" + item.Gender
                _html += "<td id='EmailId_" + item.ID + "'>" +
                _html += "<td>" + item.CreatedDate + "</td>";
                _html += "<td><div onclick='EditDetails(" + item.ID +
                _html += "<td><div onclick='DeleteDetails(" + item.ID +
                _html += "</tr>";
            });
        }
    }
});

```

```

        })
        $("#tbody_Append").append(_html);
    }
    //else {
    //    $("#tbody_Append").append("<tr><td colspan='4'>No Data
Found</td></tr>");
    //}
    }
})

```

SINGLE TON

```

public class SingletonUses
{
    private SingletonUses() { }

    private static SingletonUses instance = null;

    public static SingletonUses Instance
    {
        get
        {
            if (instance == null)
            {
                instance = new SingletonUses();
            }
            return instance;
        }
    }

    public void addMethod(int a, int b)
    {
        Console.WriteLine(a + b);
    }
    public void SubMethod(int a, int b)
    {
        Console.WriteLine(a - b);
    }
}

static void Main(string[] args)
{
    SingletonUses.Instance.addMethod(1, 5);
    SingletonUses.Instance.SubMethod(5, 1);
}

```


View Model Display Model & Update Model in mvc

```
public class Student
{
    [Key]
    public int Id { get; set; }

    [Required]
    [MaxLength(20)]
    public string Name { get; set; }

    [Required]
    [MaxLength(10)]
    public string Gender { get; set; }

    [Required]
    [MaxLength(10)]
    public string EmailId { get; set; }

    public string Ip { get; set; }
}

public class VmStudent
{
    public string Name { get; set; }
    public string Gender { get; set; }
    public string EmailId { get; set; }
    public string Ip { get; set; }
}

[HttpPost]
public ActionResult Create(FormCollection collection)
{
    try
    {
        if (ModelState.IsValid)
        {
            VmStudent VmStudent = new Models.VmStudent();
            UpdateModel(VmStudent);
            string abc = VmStudent.EmailId;
            // TODO: Add insert logic here

            return RedirectToAction("Index");
        }
        catch
        {
            return View();
        }
    }
}

@model WebApplication2.Models.Student

@using (Html.BeginForm())
{
```

```

@Html.AntiForgeryToken()

<div class="form-horizontal">
    <h4>Student</h4>
    <hr />
    @Html.ValidationSummary(true, "", new { @class = "text-danger" })
    <div class="form-group">
        @Html.LabelFor(model => model.Name, htmlAttributes: new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class = "form-control" } })
            @Html.ValidationMessageFor(model => model.Name, "", new { @class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Gender, htmlAttributes: new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.Gender, new { htmlAttributes = new { @class = "form-control" } })
            @Html.ValidationMessageFor(model => model.Gender, "", new { @class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.EmailId, htmlAttributes: new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.EmailId, new { htmlAttributes = new { @class = "form-control" } })
            @Html.ValidationMessageFor(model => model.EmailId, "", new { @class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.Ip, htmlAttributes: new { @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.Ip, new { htmlAttributes = new { @class = "form-control" } })
            @Html.ValidationMessageFor(model => model.Ip, "", new { @class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" value="Create" class="btn btn-default" />
        </div>
    </div>
</div>
}

```

```

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}

public static void Register(HttpConfiguration config)
{
    RouteTable.Routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate:
            "api/{controller}/{action}/{id}/{pageSize}/{currentPage}/{mode}",
        defaults: new
        {
            id = RouteParameter.Optional,
            pageSize = System.Web.Http.RouteParameter.Optional,
            currentPage = System.Web.Http.RouteParameter.Optional,
            mode = System.Web.Http.RouteParameter.Optional,
        }
    ).RouteHandler = new SessionRouteHandler();
    config.Formatters.Remove(config.Formatters.XmlFormatter);
    //config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new
    CamelCasePropertyNamesContractResolver();
}

public class SessionRouteHandler : System.Web.Routing.IRouteHandler
{
    System.Web.IHttpHandler
    System.Web.Routing.IRouteHandler.GetHttpHandler(System.Web.Routing.RequestContext
    requestContext)
    {
        return new SessionControllerHandler(requestContext.RouteData);
    }
}

public class SessionControllerHandler :
    System.Web.Http.WebHost.HttpControllerHandler,
    System.Web.SessionState.IRequiresSessionState
    {
        public SessionControllerHandler(System.Web.Routing.RouteData routeData)
            : base(routeData)
        { }
    }
}

```

Nowadays, Asynchronous programming is very popular with the help of the async and await keywords in C#.

When we are dealing with UI and on button click, we use a long running method like reading a large file or something else which will take a long time, in that case, the entire application must wait to complete the whole task.

In other words, if any process is blocked in a synchronous application, the entire application gets blocked and our application stops responding until the whole task completes.

Asynchronous programming is very helpful in this condition. By using Asynchronous programming, the Application can continue with the other work that does not depend on the completion of the whole task.

We will get all the benefits of traditional Asynchronous programming with much less effort by the help of async and await keywords.

Suppose, we are using two methods as Method1 and Method2 respectively and both the methods are not dependent on each other and Method1 is taking a long time to complete its task. In Synchronous programming, it will execute the first Method1 and it will wait for completion of this method and then it will execute Method2. Thus, it will be a time intensive process even though both the methods are not depending on each other.

We can run all the methods parallelly by using the simple thread programming but it will block UI and wait to complete all the tasks. To come out of this problem, we have to write too many codes in traditional programming but if we will simply use the async and await keywords, then we will get the solutions in much less code.

Also, we are going to see more examples, if any third Method as Method3 has a dependency of method1, then it will wait for the completion of Method1 with the help of await keyword.

Async and await are the code markers, which marks code positions from where the control should resume after a task completes.

Let's start with practical examples for understanding the programming concept.

Sample examples of async and await keyword in C#

We are going to take a console Application for our demonstration.

Example 1

In this example, we are going to take two methods, which are not dependent on each other.

```
1.  static void Main(string[] args)
2.  {
3.      Method1();
4.      Method2();
5.      Console.ReadKey();
6.  }
7.
8.  public static async Task Method1()
9.  {
10.     await Task.Run(() =>
11.     {
12.         for (int i = 0; i < 100; i++)
13.         {
14.             Console.WriteLine(" Method 1");
15.         }
16.     });
17. }
18.
19.
20. public static void Method2()
21. {
22.     for (int i = 0; i < 25; i++)
23.     {
24.         Console.WriteLine(" Method 2");
25.     }
26. }
```

Example 2

In this example, Method1 is returning total length as an integer value and we are passing a parameter as a length in a Method3, which is coming from Method1.

Here, we have to use await keyword before passing a parameter in Method3 and for it, we have to use the async keyword from the calling method.

We can not use await keyword without async and we cannot use async keyword in the Main method for the console Application because it will give the error given below.



We are going to create a new method as callMethod and in this method, we are going to call our all Methods as Method1, Method2 and Method3 respectively.

Code sample

```
1. class Program
2. {
3.     static void Main(string[] args)
4.     {
5.         callMethod();
6.         Console.ReadKey();
7.     }
8.
9.     public static async void callMethod()
10.    {
11.        Task<int> task = Method1();
12.        Method2();
13.        int count = await task;
14.        Method3(count);
15.    }
16.
17.    public static async Task<int> Method1()
18.    {
19.        int count = 0;
20.        await Task.Run(() =>
21.        {
22.            for (int i = 0; i < 100; i++)
23.            {
24.                Console.WriteLine(" Method 1");
25.                count += 1;
26.            }
27.        });
28.        return count;
29.    }
30.
31.
32.    public static void Method2()
33.    {
34.        for (int i = 0; i < 25; i++)
35.        {
36.            Console.WriteLine(" Method 2");
37.        }
38.    }
39.
40.
41.    public static void Method3(int count)
42.    {
43.        Console.WriteLine("Total count is " + count);
44.    }
45. }
```

In the code given above, Method3 requires one parameter, which is the return type of Method1. Here, await keyword is playing a vital role for waiting of Method1 task completion.

Real time example


There are some supporting API's from the .NET Framework 4.5 and the Windows runtime contains methods that support async programming.

We can use all of these in the real time project with the help of async and await keyword for the faster execution of the task.

Some APIs that contain async methods are HttpClient, SyndicationClient, StorageFile, StreamWriter, StreamReader, XmlReader, MediaCapture, BitmapEncoder, BitmapDecoder etc.

In this example, we are going to read all the characters from a large text file asynchronously and get the total length of all the characters.

Sample code



```
1. class Program
2. {
3.     static void Main()
4.     {
5.         Task task = new Task(CallMethod);
6.         task.Start();
7.         task.Wait();
8.         Console.ReadLine();
9.     }
10.
11.     static async void CallMethod()
12.     {
13.         string filePath = "E:\\sampleFile.txt";
14.         Task<int> task = ReadFile(filePath);
15.
16.         Console.WriteLine(" Other Work 1");
17.         Console.WriteLine(" Other Work 2");
18.         Console.WriteLine(" Other Work 3");
19.
20.         int length = await task;
21.         Console.WriteLine(" Total length: " + length);
22.
23.         Console.WriteLine(" After work 1");
24.         Console.WriteLine(" After work 2");
25.     }
26.
27.     static async Task<int> ReadFile(string file)
28.     {
29.         int length = 0;
30.
31.         Console.WriteLine(" File reading is stating");
32.         using (StreamReader reader = new StreamReader(file))
33.         {
34.             // Reads all characters from the current position to the end of the stream asynch
35.             // and returns them as one string.
36.             string s = await reader.ReadToEndAsync();
37.
38.             length = s.Length;
39.         }
40.         Console.WriteLine(" File reading is completed");
41.         return length;
42.     }
43. }
```

In the code given above, we are calling a ReadFile method to read the contents of a text file and get the length of the total characters present in the text file.

In our sampleText.txt, file contains too many characters, so It will take a long time to read all the characters.

Here, we are using async programming to read all the contents from the file, so it will not wait to get a return value from this method and execute the other lines of code but it has to wait for the line of code given below because we are using await keyword and we are going to use the return value for the line of code given below..

```
1. int length = await task;
2. Console.WriteLine(" Total length: " + length);
```

Subsequently, other lines of code will be executed sequentially.

```
1. Console.WriteLine(" After work 1");
2. Console.WriteLine(" After work 2");
```

Set Cookies

```
HttpCookie StudentCookies = new HttpCookie("StudentCookies");
StudentCookies.Value = "hallo";
StudentCookies.Expires = DateTime.Now.AddMinutes(1);
Response.SetCookie(StudentCookies);
```

Read Cookies

```
if
(this.ControllerContext.HttpContext.Request.Cookies.AllKeys.Contains("StudentCookies"))
    abc = Convert.ToString(Request.Cookies["StudentCookies"].Value);
```

```
declare @value1 nvarchar(100);
```

```
create table #tempStudent (f_Name nvarchar(100));
declare Pramcur cursor for select f_Name from t_BVR_Students
```

```
open Pramcur
```

```
fetch next from Pramcur into @value1
```

```
while @@FETCH_STATUS =0
begin
```

```
    insert into #tempStudent values(@value1)
```

```
    fetch next from Pramcur into @value1
```

```
end
```

```
close Pramcur
```

```
deallocate Pramcur
```

```
select * from #tempStudent
```

Set Data

```
ViewBag.MilnaaUserId = "1023";
ViewData["MyUserId"] = "1024";
TempData["YourUserID"] = "1025";
//TempData.Peek("YourUserID");
```

Read Data

```
string Abc = Convert.ToString(ViewBag.MilnaaUserId);
string obj = Convert.ToString(ViewData["MyUserId"]);
string syz = Convert.ToString(TempData["YourUserID"]);
```

```
create table #tempTempo (f_Id bigint identity(1,1), f_Name nvarchar(100) UNIQUE)

declare @Tablvar table
(
f_Name nvarchar(200)
)

insert into @Tablvar values(@Name)

select * from @Tablvar
```

Let us compile the list for differences.

⇒ Table variable (@table) is created in the memory. Whereas, a Temporary table (#temp) is created in the tempdb database. However, if there is a memory pressure the pages belonging to a table variable may be pushed to tempdb.

⇒ Table variables cannot be involved in transactions, logging or locking. This makes @table faster than #temp. So table variable is faster than temporary table.

⇒ Temporary tables are allowed CREATE INDEXes whereas, Table variables aren't allowed CREATE INDEX instead they can have index by using Primary Key or Unique Constraint.

⇒ Table variable can be passed as a parameter to functions and stored procedures while the same cannot be done with Temporary tables.

⇒ Temporary tables are visible in the created routine and also in the child routines. Whereas, Table variables are only visible in the created routine.

⇒ Temporary table allows Schema modifications unlike Table variables

But to answer your question, there is not a size limit to viewstate.

Delete Duplicate Records

```
with CTE
as
(select ROW_NUMBER() over(partition by f_Name order by f_Name ) as RowNum, * from
t_BVR_Students )
delete from CTE where RowNum >1
```