بسمه تعالی

دانشگاه آزاداسلامی واحد لاهیجان

دستور کار آزمایشگاه پایگاه داده ها

مدرس : حسن – علی اکبرپور
HAS@TAVANA.NET
H_ALIAKBARPOUR@YAHOO.COM

دستور کار آزمایشگاه پایگاه داده ها

آزمایش 1

آشنایی با محیط QUERY ANALYZER ، نحوه ایجاد پایگاه داده وجداول پیش آگاهی

مقدمه :آشنایی با RDBMSها

از زمانی که در سال 1970مقاله آقای کاد تحت عنوان "مدل رابطه ای داده ها برای بانک های اطلاعاتی اشتراکی بزرگ" منتشر شد ، زمان زیادی نمی گذرد. بعد از مطرح شدن این ایده پروژه تحقیقاتی تحت عنوان SYSTEM**/R** در IBM شکل گرفت که حاصل آن ایجاد اولین DBMS وزبان **SQL** بود. از آن زمان شرکت های مختلفی به **تولید**DBMS هاپرداختندتا این که در سال1988، SQL-SERVERمعرفی شد. علاوه بر این در سال 1986، **SQL** توسط **ANSI** استاندارد شد. نسخه ای از این زبان را ، تحت عنوان TRANSACT-**SQL** استفاده می نماید.
همان گونه که می دانید زبان **SQL** غیررویه ای است یعنی در آن تنها درخواست کاربرارائه می گرددوالگوریتم لازم برای اجرای آن توسط بخش هایی از**DBMS**(بهینه ساز) تولید می گردد .
هرسیستم مدیریت پایگاه داده ای بر مبنای مدل رابطه ای ،یک **RDBMS** نامیده می شود .این سیستم هاازدو بخش عمده به شرح زیرساخته شده اند**:**

1- هسته :که کارهای مدیریتی را انجام می دهد.

2- فرهنگ داده ها(**DataDictionary**) : که شامل اطلاعاتی در مورد عناصرواشیاء مختلف پایگاه داده ای تعریف شده می باشد ازقبیل**sysobjects**که مشخصات اشیاءمختلف تعریف شده درآن نگهداری می گردد یا**sysindexes** و**syscolumns** که مشخصات شاخص هاومشخصات ستون های تعریف شده در آن ها ذکر می گردد.

آشنایی با SQL-SERVER به عنوان یک RDBMS

برای راهبری پایگاه داده ها، **SQL-SERVER** دو واسط گرافیکی به نام های **Enterprise Manager** و**Query Analyzer** را در اختیار قرار می دهد. دراین مرحله فرض می کنیم یک سرور ثبت شده وتعدادی شی **Login** وتعدادی پایگاه داده تعریف شده برروی آن داریم بنابراین ازمباحث مربوط به آنها فعلا چشم می پوشیم . علاوه بر این فرض می کنیم که سرور فوق در حال سرویس دهی است.**Query Analyzer** اولین واسطی است که به کمک آن می توانید پرس وجو های **T-SQL**وتوابع و روال های ذخیره شده را اجرا کنید . پس ازورودبه**Query Analyzer** اطلاعات **Login** از شما خواسته می شود که می توانید از تایید اعتبار خود ویندوز استفاده کنیدو به این محیط وارد شوید. این محیط از دوپنجره اصلی تشکیل شده است. درپنجره سمت چپ با نام **Object Browser** می توانید اشیایی از قبیل نام سروری که به آن متصل شده اید، پایگاه داده های موجود وبقیه اشیاء اصلی مورد استفاده در**SQL-SERVER** را مشاهده وویرایش کنید. در این قسمت اشیاء به صورت ساختار درختی نمایش داده می شوند. در هرسرور موجود حداقل چهارپایگاه داده زیر موجود است :

1-**Master** : شامل تمامی اطلاعات لازم برای مدیریت پایگاه، مانند پایگاه داده های تعریف شده ومشخصات کاربران و رویه های ذخیره شده سیستمی وپیام های خطاست. جداولی که در این پایگاه داده موجود است معمولاکاتالوگ سیستم نامیده می شود .

2- **Model** : یک الگو برای ساخت پایگاه داده های جدید است وهر شی موجود در آن ،در پایگاه داده های جدید ایجاد می شود. برای مثال اگر یک شناسه کاربر جدید در آن قرار دهید ،در تمامی پایگاه هایی که بعد ازاین ایجاد می شوند این شناسه هم وجود دارد .

3-**Msdb** : در نگهداری برنا مه های زمان بندی سیستم و**Job** ها وتاریخچه نسخه های پشتیبان کاربرد دارد .

4-**Tempdb** : محل موقتی برای اشیائی است که نیاز به فضای موقتی دارند .

در زیر پنجره**object browser** ،با کلیک بر روی تب **Templates** می توانید به **Template** های موجود در مورد هر شی دسترسی داشته باشید . پنجره دیگر موجود در محیط **Query Analyer**پنجره پرس وجو است که از آن برای اجرای پرس وجو های تعاملی استفاده می شود .

برای اجرای اسکریپت ها (مجموعه هایی از دستورات) باید یکی از پایگاه های داده ای موجود به عنوان پایگاه داده جاری انتخاب شود. برای این کار از دستور**USE**، استفاده می شود. این پنجره از دو قسمت تشکیل شده است، که یکی برای ویرایش دستورات و دیگری برای نمایش نتایج به کار می رود. (قسمت اخیر بعداز اجرای یک اسکریپت قابل مشاهده است.)کاربر می تواند با استفاده از گزینه**Query**، نحوه نمایش این خروجی ها را به حالت **Grid**یا **Text**تنظیم کند یا یک فایل را به عنوان محل ذخیره خروجی های **Query** تعریف نماید .

به ثبت رساندن وحذف وتغییر یک پایگاه داده جدید درSQL-SERVER

اصولا اطلاعات موجود در هر پایگاه داده در فایل ها ی آن پایگاه داده نگهداری می شوند. این امکان وجود دارد که فایل های مختلف ،گروه های مختلفی را تشکیل دهند که هر فایل به یکی از آن ها اختصاص داشته باشد در این صورت می توان اشیاء پایگاه داده را در یک فایل خاص یا فایل های یک گروه ذخیره کرد. علاوه بر این مثلا دراعمالی مانند پشتیبانی (**backup**) می توان به جای کل پایگاه داده ای، گروه های فایل اصلی را پشتیبانی کردیا بااعمال برنامه زمانی مورد نظر، هر یک از گروه هارامستقلاپشتیبانی کرد. برای پایگاه داده های تعریف شده در **SQL-SERVER** سه نوع فایل قابل تصور است :

1- فایل های **Primary**(باپسوند**mdf.**)**:** که حاوی اطلاعات راه اندازی پایگاه هستند وبه بقیه فایل های پایگاه داده ها اشاره دارند .

2- فایل های **Secondary**(باپسوند**ndf.**) : بقیه فایل های داده ای به جز فایل های داده ای اصلی در این گروه قرارمی گیرند .

3- فایل های **Log**(باپسوند**ldf.**) : برای ثبت تراکنش های موجود در پایگاه به کار می روندو عضو هیچ گروه فایلی نیستند. برای بسیاری از پایگاه های داده ای معرفی گروه 1و3کافی است وممکن است پایگاه داده ای چندین فایل از نوع دوم داشته باشد یا هیچ فایلی از این نوع نداشته باشد. یک فایل نمی تواند عضو بیش ازیک گروه فایلی باشد وفایلهای سیستم در گروه فایل **Primary** قرار می گیرند. (برای اطلاعات بیشتر در مورد نحوه استفاده از این امکان در عمل می توانید به مراجع **SQL-SERVER**مراجعه کنید .)

ایجاد جداول پایگاه داده

این جداول به منظور تعریف فیلدهای مورد استفاده در ایجاد یک پایگاه داده ای مورد استفاده قرار می گیرند که در قسمت پیوست ساختار کلی یک جدول آورده شده است .
درهرجدول پایگاه داده ای برای اطمینان از درستی مقادیر فیلدها انواع جامعیت داده ای، مورداستفاده قرار می گیرد که ازانواع آن می توان به موارد زیر اشاره کرد :
کلیداصلی**(primarykey)** :که شامل یک یاچندستون است که مقادیرموجود دردوسطر از ستون ها نمیتوانند یکسان باشند. همچنین کلید اصلی نمی تواند مقدارتهی بپذیرد .
کلید کاندیدا**(unique)** : مانند کلیداصلی است با این تفاوت که درستون های تعریف شده به عنوان کلید کاندیدا می تواند مقدار تهی هم وارد شود .
کلید خارجی**(foreignkey)** : برای ایجاد ارتباط بین داده های جداول پایگاه داده ای ، ازیک یا ترکیبی از چند ستون با عنوان کلید خارجی استفاده می شود به طوری که داده های یک جدول با مقادیرکلید اصلی جدول مرتبط با آن پرمی شود.
Check : مقادیر قابل پذیرش یک ستون توسط این محدودیت کنترل می شود .
Constraints : برای اعمال محدودیت های داده ای ، مانند کلیداصلی ،کلید خارجی و کلید کاندیدا و**check** مورد استفاده قرار می گیرد .

ستون های محاسباتی در تعریف جداول

این ستون ها عبارت به کار رفته برای محاسبه داده را به جای خودداده ذخیره می کنند و قواعد زیر را دارند :

- ستون های ارجاع شده در عبارت ستون محاسباتی باید در همان جدول باشند .
- ستون محاسباتی شامل **subquery** نیست .
- این ستون ها به عنوان جزئی از کلید یا اندیس به کار نمی روند .
- نمی توانند محدودیتی از نوع **default** داشته باشند .(چرا؟)
- در دستورات **insert** و**update** ارجاعی به آن ها نداریم .

مثال1 :

```
Create table author
(au_id1  int primary key clustered,
 au_id2  int  unique nonclustered,
 au_degree smallint,
 au_name char(16) not null,
 au_family char(16) not null,
 au_address char(30) null)
 CONSTRAINT au_degree_chk check((au_degree>=0)
and(au_degree<=100)
```

در جدول تعریف شده بالا ،برای هر نویسنده دو شماره ازنوع **int** تعریف شده که روی اولی شاخص خوشه ای و روی دومی شاخص غیرخوشه ای قرارگرفته است. علاوه بر این روی شماره دوم محدودیت یکتا بودن اعمال می شود. سومین ستون نیز به درجه نویسنده اشاره می کند که با **check** روی آن محدودیت بازه ای اعمال می شود .

جدول های موقتی

این جداول زمانی که اتصال به سرورقطع شود از بین می روند. مهم ترین انواع جدول های موقتی مانند متغیرها انواع عمومی و محلی هستند. تفاوت این دو نوع در این است که ازجداول سراسری همه کاربران سیستم می توانند استفاده کنندو با ## شروع می شوند. جداول محلی فقط برای کاربر سازنده خود قابل استفاده هستند و با # شروع می شوند. برای ایجاد این جداول از دستور**CREATE** استفاده می شود :

```
CREAET TABLE #myTmpTable
(Name VARCHAR(30) NOT NULL
ADDRESS VARCHAR(50))
```

از دستور **SELECT ---INTO** می توان برای تعریف و پرکردن یک جدول موقتی به صورت همزمان استفاده کرد .

```
Select Name,Cname       INTO #myTmpTable
From STD AS S

INNER JOIN
        CRS AS C
        ON C.S#=S.S#
```

شاخص ها در **SQL-Server** اشیاء خاصی هستند که این امکان را فراهم می آورند که بتوان بر اسا س مقادیر یک یا چند ستون به سرعت به سطر های یک جدول دسترسی پیدا کرد. **SQL-Server**، دو نوع شاخص**Clustered** و **Nonclustered** رادر اختیار کاربر برای تعریف قرار می دهد .**SQL-Server** برای پیاده سازی شاخص ها از ساختار **B-TREE** استفاده می کند که در آن برگ ها حاوی داده های واقعی هستند .

1- شاخص **Clustered** : دراین نوع شاخص ، داده ها واقعا از نظر فیزیکی مرتب می شوند. در این نوع اندیس برگ ها داده های واقعی هستند . همیشه با معرفی یک ستون به عنوان کلید اصلی به صورت خودکار روی آن یک شاخص **Clustered** ساخته می شود .

2- شاخص **Nonclustered** : در این نوع شاخص برگ ها بر مبنای ستون هایی که به عنوان شاخص تعریف شده اند مرتب می شوند ولی تفاوت آن با نوع قبلی این است که در صورتی که قبلا روی جدول شاخص **Clustered** ایجاد نشده باشد برگ های اندیس **Nonclustered** حاوی آدرس ذخیره تاپل(ونه خود تاپل) متناسب با کلید شاخص شده است ودرغیراین صورت حاوی مقدار کلید شاخص دارمتناسب با آن تاپل هستند .

با توجه به توضیحات بالا رعایت نکات زیر توصیه می شود :

- برای ستون دارای شاخص **Clustered** از مقادیرحجم داراستفاده نکنید. زیرا علاوه بر اینکه زمان جستجو در ستون با شاخص **Clustered**افزایش می یابد ، حجم ذخیره سازی شاخص های **Nonclustered** هم افزایش می یابد. (چرا ؟)

نکته: سعی کنید ابتدا شاخص **Clustered**را ایجا دکنید و بعد شاخص های **Nonclustered** راتعریف کنید . (چرا ؟)

تعریف شاخص:

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX
index_name
   ON { table | view } ( column [ ASC | DESC ] [ ,...n ] )
[ ON filegroup ]
```

توضیح بعضی پارامترهای تعریف بالا:

- کلمه کلیدی **Unique** مشخص می کندکه مقادیر تکراری در شاخص غیرمجازند .
- **ASC** بیان کننده این است که می خواهیم شاخص به صورت صعودی ساخته شود .**DESC** نیز درخواست
  نزولی بودن شاخص را بیان میکند .
- با استفاده از گزینه**ON filegroup** می توان یکی از گروه های فایلی را به عنوان محل ذخیره شاخص معرفی کرد .

مثال**2** : فرض کنید می خواهیم شاخصی با نام **Snumindex** را روی ستون شمار ه دانشجویی به صورت نزولی در جدول **STD** ایجاد کنیم :

```
Create  clustered index Snumindex on STD(S# DESC )
On Primary
```

سوال : فرض کنید درخواست های زیادی مطرح می شود که در آن ها معمو لا در گزینه **Where** سوال روی شماره دانشجویی های پایین مطرح می شود آیاحذف کردن این اندیس می تواند زمان پاسخگویی ر ابه طور نسبی افزایش دهد؟

تغییر یک جدول

```
ALTER TABLE author
ADD column_b  INT  IDENTITY      اضافه کردن یک ستون با محدودیت کلید اصلی
 CONSTRAINT column_b_pk PRIMARY KEY,

/* Add a column with a constraint to enforce that   */
/* nonnull data is in a valid phone number format.  */
column_d VARCHAR(16) NULL
CONSTRAINT column_d_chk
CHECK
(column_d IS NULL OR
 column_d LIKE '[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]'OR
 column_d LIKE
 ('[0-9][0-9][0-9]) [0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]'),

/* Add a nonnull column with a default.  */
```

column_e DECIMAL(3,3)
CONSTRAINT column_e_default
DEFAULT .081

توجه : برای انجام آزما یش ها شکل دستورات **create database** ، **Alter database** ، **create table** ، **Alter table** و انواع داده ای موجود در ضمیمه را مطالعه کنید.

دستور کار:

بخش اول-آشنایی با **Query Analyzer** ونحوه اجرای دستورات **T-SQL**

1- بعد از **login**کردن با نام عبوری که به شما داده می شود و روی سروری که به شما معرفی می شود وارد محیط **Query Analyzer** شوید .

2- روی سروری که به آن **Login**کرده اید کلیک کنید وپایگاه داده های موجود روی این سرور را مشاهده کنید .چه پایگاه داده هایی روی این سرور به ثبت رسیده است؟

3- به کمک دستور**exec** رویه های ذخیره شده **sp_help** (برای پایگاه داده **pubs**) و **sp_help db**را اجرا کنید . در پنجره **results** خروجی هایی هر کدام را مشاهده می کنید . در مورد عملکرد آن ها توضیح دهید؟

(این دوازرویه های ذخیره شده **(stored procedure)** سیستمی هستند که بعدا با نحوه ایجاد ومدیریت این رویه ها آشنا می شوید .)

4- الف- اسکریپت زیر را به یکباره اجرا کنید ویک فایل را به عنوان محل ذخیره خروجی ها تعریف کنید . (دسته مجموعه ای از دستورات **T-SQL** است که همگی به یکباره به موتور پایگاه ارسال می شوند و کامپایل وبهینه سازی و اجرا می شوند**.** اسکریپت نیزمجموعه ای از یک یا چند دسته است که به صورت گروهی ذخیره می شوند. از اسکریپت ها برای مثلا بخشی از کار بارگذاری داده ها (**Data overloading**)یا نگهداری پایگاه داده استفا ده می شود. مثلا اگر نیاز به انجام چندکارمستقل باشد، اسکریپتی از چنددسته می نویسیم و برای مشخص کردن دسته ها از هم از **Go**در بین آنها استفاده می کنیم. بدین ترتیب خطاهای روی هر دسته اعم از کامپایل یاخطاهای زمان اجرا ، اجرای دسته های قبلی وبعدی را مختل نمی کند .)

```
Use pubs
Select *  from authors
Go
Use Northwind
Select EmployeeID from Employees
where City='London'
```

ب- در دسته دوم **EmployeeID** را به**Employee** تغییردهیدواین اسکریپت رااجرا کنید . آیاتغییر یک دسته دراجرای کل اسکریپت تأثیر گذار است ؟ توضیح دهید.

بخش دوم - ایجاد یک پایگاه داده جدید

1- اسکریپتی برای ایجاد پایگاه داده ای بانام **Test** با مشخصات زیر اجرا کنید :

- یک فایل اصلی با مشخصات اندازه**MB** 20 باحداکثررشد مساوی **100MB**و با رشد **2MB**درهربارودر گروه فایل اصلی
- یک فایل ثانویه با مشخصات اندازه **5MB**حداکثر اندازه **10MB**وبا رشد **1MB**و در گروه فایلی با نام **test** واین گروه فایلی را گروه فایلی پیش فرض قرار دهید .
- یک فایل ثبت تراکنش**(log file)** با مشخصات اندازه **10MB** حداکثر اندازه **30MB**وبا رشد **20%**

2- رویه های ذخیره شده**sp_helpfilegroup** و**sp_helpfile** را روی پایگاه داده ای که ایجاد کرده اید اجرا کنید . چه خروجی هایی مشاهده می کنید ؟

3- اسکریپتی را اجرا کنید که گروه فایلی **test** وفایل های آن را حذف کند . (اصولا برای حذف هر گروه فایلی ابتدا گروه فایلی دیگری رابه عنوان گروه فایل پیش فرض تعریف میکنیم در صورتی که گروه فایلی حذف شوند پیش فرض باشد سپس تمام فایل های آن گروه فایلی وبعد خودآن را حذف می کنیم .)

4- پایگاه داده فوق را به **registration** تغییر نام دهید .(از رویه **sp_renamedb** استفاده کنید .)

5- کلیه جداول پایگاه داده **registration** راکه درزیرمشخص شده، باتوجه به محدودیت های موردنظرتعریف کنید .(کلیه اسکریپت های نوشته شده را ذخیره کنید .)

جداول موردنیازدرآزمایشگاه :

فایل اطلاعات دانشجو (**STD**)						
نام لاتین	نام فارسی	نوع داده ای	امکان **null**	کلید اصلی	کلید خارجی و ارجاع	قیود
S#	شماره دانشجویی	**Int**		×		1- ازنوع **clustered**تعریف شود، 2- رقم اول هرشماره از1تا9 ،بقیه ارقام از0تا9
Name	نام	**Varchar(16)**				
Family	فامیل	**Varchar (20**				
Field	رشته تحصیلی	**Tinyint**				رقم اول بین1تا9 ،رقم دوم بین0تا9
Sex	جنسیت **(F-M)**	**Char(1)**				مرد:**M** , زن:**F** **Check (sex='M' or sex='f')**
Gpa	معدل کل	**Dec(5,2)**	×			

Address	آدرس	**Varchar(40)**	×			
Citycode	کدشهر	**Int**				
TelNo	شماره تلفن	**Int**	×			رقم اول هرشماره از 1تا9 ،بقیه ارقام از 0تا9
Ssno	شماره شناسایی ملی	**Bigint**				1-ازنوع **unique** تعریف شود، **2-**رقم اول هرشماره از 1تا9 ،بقیه ارقام از 0تا9
Birthdate	تاریخ تولد	**Datetime**				

فایل درس (**CRS**)						
نام لاتین	نام فارسی	نوع داده ای	امکان **null**	کلید اصلی	کلید خارجی و ارجاع	قیود
C#	شماره درس	**Char(7)**		×		رقم اول هرشماره از 1تا9 ،بقیه ارقام از 0تا9
Cname	نام درس	**Varchar (30)**				
Unit	تعداد واحد درس	**Dec(2,1)**				
Passgrade	حداقل نمره قبولی دردرس	**Dec(5,2)**				**Check (0=<passgrade <=20)**
Crstype	نوع درس (تئوری – عملی)	**Char (1)**				تئوری=**'t'**وعملی=**'p'** **Check(Crstype='t' or crstype ='p')**

فایل ترم دانشجو (**STDTRM**)						
نام لاتین	نام فارسی	نوع داده ای	امکان **null**	کلید اصلی	کلیدخارجی و ارجاع	قیود
TrmNo	شماره ترم	**Char(4)**		×		رقم اول بین3تا9 ،رقم دوم وسوم بین0تا9 ورقم چهارم بین1تا3
S#	شماره دانشجویی	**Int**		×	**STD(S#)**	
TrmGpa	معدل ترم	**Dec(5,2)**	×			

فایل پیش نیاز (PREREQ)						
نام لاتین	نام فارسی	نوع داده ای	امکان **null**	کلید اصلی	کلید خارجی و ارجاع	قیود
C#	شماره درس	**Char(7)**		×	**CRS(c#)**	
Cp#	شماره درس پیش نیاز	**Char(7)**		×	**CRS(c#)**	
SeqNo	شماره چندمین پیش نیازیا هم نیاز	**Tiny int**		×		**Check(1=<seqno <=5)**

فایل ثبت نام (REG)						
نام لاتین	نام فارسی	نوع داده ای	امکان **null**	کلید اصلی	کلید خارجی و ارجاع	قیود
TrmNo	شماره ترم	**Char(4)**		×	**STDTRM(TRMNO)**	رقم اول بین3تا9 ،رقم دوم وسوم بین0تا9 ورقم چهارم بین1تا3
S#	شماره دانشجویی	**Int**		×	**STD(s#)** **STDTRM(S#)**	
C#	شماره درس	**Char(7)**		×	**CRS(c#)**	
Grade	نمره دردرس	**Dec(5,2)**	×			**Check(0=<Grade<=20)**

فایل هم نیاز (COREQ)						
نام لاتین	نام فارسی	نوع داده ای	امکان **null**	کلید اصلی	کلید خارجی و ارجاع	قیود
C#	شماره درس	**Char(7)**		×	**CRS(c#)**	
Cc#	شماره درس هم نیاز	**Char(7)**		×	**CRS(c#)**	
SeqNo	شماره چندمین پیش نیاز یا هم نیاز	**Tiny int**		×		**Check(1=<seqno<=5)**

فایل کد (Code)						
نام لاتین	نام فارسی	نوع داده ای	امکان null	کلید اصلی	کلید خارجی و ارجاع	قید
Field	رشته تحصیلی	Varchar(8)		×		
Type	نوع	Varchar(4)		×		
Desc	شرح	Varchar(30)		×		

سوال :

1- آیا روی هر جدول محدودیتی روی تعداد ایندکس های قابل تعریف وجود دارد؟

2-آیا ساخت فهرست به طور نامحدود مشکلی برای سیستم (ازنظر زمان عملیات) روی هریک ازعملیات **select,insert,delete,update** ایجاد می کند؟

بخش سوم – آشنایی با نحوه حذف یک پایگاه داده وجداول

برای حذف اشیاءیک پایگاه داده از دستور **drop** استفاده می کنیم.

۱- پایگاه داده جدیدی با نام **test**تعریف کرده وجدول مثال زده شده در قسمت پیش مطالعه را برای آن تعریف کنید .

۲- با استفاده از دستور **Alter table** ستون نگهدارنده **SSNO** راازجدول **STD**) حذف کنید .با چه خطایی برخورد می کنید .چرا؟

۳- بعد از حذف محدودیت ایندکس گذاشته شده بر روی **SSNO** ،خود **SSNO**راحذف کنید .

۴- اطلاعات دانشجویی مجازی را در جدول **STD**وارد کنید .

۵- اطلاعات دروس مجازی با شماره های 1024345و1025123و**86**102را در جداول **CRS** و**PREREQ**و**COREQ**وارد کنید . (فرض کنید درس 1024345پیش نیازدرس 1025123وهم نیاز درس 102686 است .)

۶- سعی کنید درس 1027456را برای تنها دانشجوی موجود در جدول **STD**در ترم 3832ثبت نام کنید .آیا این کار امکان پذیر است چرا؟ با این کار کدام یک از قواعد جامعیت داده ها نقض می شود؟ آیا در این مرحله امکان ثبت نام درس1025123 برای این دانشجو امکان دارد ؟

۷- ستون حذف شده **SSNO**ر ابه جدول **STD**با محدویت کلید ثانویه یا ایندکس غیر خوشه ای بودن و غیر قابل تهی بودن برای این جدول تعریف کنید ؟با چه خطایی برخورد می کنید؟

۸- راهی برای انجام عمل فوق بی‌ابی‌د.

۹- حال با استفاده از دستور **update** یک شماره شناسایی ملی برای دانشجوی مورد نظر وارد کنید .

۱۰- درس شماره 1024345را برای دانشجوی فوق ثبت نام کنید؟

۱۱- در این مرحله سعی کنید جدول **STD**را**drop**کنید ؟آیا این کار امکان پذیر است؟چرا؟

۱۲- تمامی رکوردهای موجوددرجداول **STD**و **REG**راحذف کنید؟ این کار به چه ترتیبی باید انجام شود؟

13- با استفاده از فرمان **Alter table**دو ستون با عنوان های **Totpassunit**و**Totregunit**که به ترتیب نگهدارنده تعداد کل واحد گذرانده وتعداد کل واحد اخذ شده هستند را به جدول **STD**اضافه کنید .

ضمیمه آزمایش اول:

تمامی مطالب این ضمیمه بر گرفته از **Sql-Server 2000 Online books**می باشد.در صورت نیاز می توانید به این منبع هم مراجعه کنید.

1-CREATE DATABASE

Creates a new database and the files used to store the database

Syntax

```
CREATE DATABASE database_name
[ ON
   [ < filespec > [ ,...n ] ]
   [ , < filegroup > [ ,...n ] ]
]
[ LOG ON { < filespec > [ ,...n ] } ]

< filespec > ::=

[ PRIMARY ]
( [ NAME = logical_file_name , ]
   FILENAME = 'os_file_name'
   [ , SIZE = size ]
   [ , MAXSIZE = { max_size | UNLIMITED } ]
   [ , FILEGROWTH = growth_increment ] ) [ ,...n ]

< filegroup > ::=

FILEGROUP filegroup_name < filespec > [ ,...n ]
```

Arguments

database_name

Is the name of the new database. Database names must be unique within a server and conform to the rules for identifiers. *database_name* can be a maximum of 128 characters, unless no logical name is specified for the log. If no logical log file name is specified, Microsoft® SQL Server™ generates a logical name by appending a suffix to *database_name*. This limits

database_name **to 123 characters so that the generated logical log file name is less than 128 characters.**

ON

Specifies that the disk files used to store the data portions of the database (data files) are defined explicitly. The keyword is followed by a comma-separated list of <filespec> items defining the data files for the primary filegroup. The list of files in the primary filegroup can be followed by an optional, comma-separated list of <filegroup> items defining user filegroups and their files.

n

Is a placeholder indicating that multiple files can be specified for the new database.

LOG ON

Specifies that the disk files used to store the database log (log files) are explicitly defined. The keyword is followed by a comma-separated list of <filespec> items defining the log files. If LOG ON is not specified, a single log file is automatically created with a system-generated name and a size that is 25 percent of the sum of the sizes of all the data files for the database.

All databases have at least a primary filegroup. All system tables are allocated in the primary filegroup. A database can also have user-defined filegroups. If an object is created with an ON *filegroup* clause specifying a user-defined filegroup, then all the pages for the object are allocated from the specified filegroup. The pages for all user objects created without an ON *filegroup* clause, or with an ON DEFAULT clause, are allocated from the default filegroup. When a database is first created the primary filegroup is the default filegroup. You can specify a user-defined filegroup as the default filegroup using ALTER DATABASE:

```
ALTER DATABASE database_name MODIFY FILEGROUP
filegroup_name DEFAULT
```

Each database has an owner who has the ability to perform special activities in the database. The owner is the user who creates the database. The database owner can be changed with sp_changedbowner.

To display a report on a database, or on all the databases for an instance of SQL Server, execute sp_helpdb. For a report on the space used in a database, use sp_spaceused. For a report on the filegroups in a database use sp_helpfilegroup, and use sp_helpfile for a report of the files in a database.

Permissions

CREATE DATABASE permission defaults to members of the sysadmin and dbcreator fixed server roles. Members of the sysadmin and securityadmin fixed server roles can grant CREATE DATABASE permissions to other logins. Members of the sysadmin and dbcreator fixed server role can add other logins to the dbcreator role. The CREATE DATABASE permission must be explicitly granted; it is not granted by the GRANT ALL statement.

CREATE DATABASE permission is usually limited to a few logins to maintain control over disk usage on an instance of SQL Server.

Examples

A. Create a database specifying multiple data and transaction log files

This example creates a database called Archive with three 100-MB data files and two 100-MB transaction log files. The primary file is the first file in the list and is explicitly specified with the PRIMARY keyword. The transaction log files are specified following the LOG ON keywords. Note the extensions used for the files in the FILENAME option: .mdf is used for primary data files, .ndf is used for the secondary data files, and .ldf is used for transaction log files.

```
USE master
GO
CREATE DATABASE Archive
ON
PRIMARY ( NAME = Arch1,
      FILENAME = 'c:\program files\microsoft sql
server\mssql\data\archdat1.mdf',
      SIZE = 100MB,
      MAXSIZE = 200,
      FILEGROWTH = 20),
( NAME = Arch2,
   FILENAME = 'c:\program files\microsoft sql
server\mssql\data\archdat2.ndf',
   SIZE = 100MB,
   MAXSIZE = 200,
   FILEGROWTH = 20),
( NAME = Arch3,
   FILENAME = 'c:\program files\microsoft sql
server\mssql\data\archdat3.ndf',
   SIZE = 100MB,
   MAXSIZE = 200,
   FILEGROWTH = 20)
LOG ON
( NAME = Archlog1,
   FILENAME = 'c:\program files\microsoft sql
server\mssql\data\archlog1.ldf',
   SIZE = 100MB,
   MAXSIZE = 200,
   FILEGROWTH = 20),
( NAME = Archlog2,
   FILENAME = 'c:\program files\microsoft sql
server\mssql\data\archlog2.ldf',
   SIZE = 100MB,
   MAXSIZE = 200,
   FILEGROWTH = 20)
GO
```

B. Create a database without specifying SIZE

This example creates a database named products2. The file prods2_dat becomes the primary file with a size equal to the size

of the primary file in the model database. The transaction log file is created automatically and is 25 percent of the size of the primary file, or 512 KB, whichever is larger. Because MAXSIZE is not specified, the files can grow to fill all available disk space.

```
USE master
GO
CREATE DATABASE Products2
ON
( NAME = prods2_dat,
   FILENAME = 'c:\program files\microsoft sql
server\mssql\data\prods2.mdf' )
GO
```

C. Create a database with filegroups

This example creates a database named sales with three filegroups:

- **The primary filegroup with the files Spri1_dat and Spri2_dat. The FILEGROWTH increments for these files is specified as 15 percent.**
- **A filegroup named SalesGroup1 with the files SGrp1Fi1 and SGrp1Fi2.**
- **A filegroup named SalesGroup2 with the files SGrp2Fi1 and SGrp2Fi2.**

```
CREATE DATABASE Sales
ON PRIMARY
( NAME = SPri1_dat,
   FILENAME = 'c:\program files\microsoft sql
server\mssql\data\SPri1dat.mdf',
   SIZE = 10,
   MAXSIZE = 50,
   FILEGROWTH = 15% ),
( NAME = SPri2_dat,
   FILENAME = 'c:\program files\microsoft sql
server\mssql\data\SPri2dt.ndf',
   SIZE = 10,
```

```
   MAXSIZE = 50,
   FILEGROWTH = 15% ),
FILEGROUP SalesGroup1
( NAME = SGrp1Fi1_dat,
   FILENAME = 'c:\program files\microsoft sql
server\mssql\data\SG1Fi1dt.ndf',
   SIZE = 10,
   MAXSIZE = 50,
   FILEGROWTH = 5 ),
( NAME = SGrp1Fi2_dat,
   FILENAME = 'c:\program files\microsoft sql
server\mssql\data\SG1Fi2dt.ndf',
   SIZE = 10,
   MAXSIZE = 50,
   FILEGROWTH = 5 ),
FILEGROUP SalesGroup2
( NAME = SGrp2Fi1_dat,
   FILENAME = 'c:\program files\microsoft sql
server\mssql\data\SG2Fi1dt.ndf',
   SIZE = 10,
   MAXSIZE = 50,
   FILEGROWTH = 5 ),
( NAME = SGrp2Fi2_dat,
   FILENAME = 'c:\program files\microsoft sql
server\mssql\data\SG2Fi2dt.ndf',
   SIZE = 10,
   MAXSIZE = 50,
   FILEGROWTH = 5 )
LOG ON
( NAME = 'Sales_log',
   FILENAME = 'c:\program files\microsoft sql
server\mssql\data\salelog.ldf',
   SIZE = 5MB,
   MAXSIZE = 25MB,
   FILEGROWTH = 5MB )
GO
```

2-ALTER DATABASE

Adds or removes files and filegroups from a database. Can also be used to modify the attributes of files and filegroups, such as changing the name or size of a file. ALTER DATABASE provides the ability to change the database name, filegroup names, and the logical names of data files and log files.

Syntax

ALTER DATABASE *database*
{ ADD FILE < filespec > [,...*n*] [TO FILEGROUP *filegroup_name*]
| ADD LOG FILE < filespec > [,...*n*]
| REMOVE FILE *logical_file_name*
| ADD FILEGROUP *filegroup_name*
| REMOVE FILEGROUP *filegroup_name*
| MODIFY FILE < filespec >
| MODIFY NAME = *new_dbname*
| MODIFY FILEGROUP *filegroup_name* {*filegroup_property* | NAME = *new_filegroup_name* }
| SET < optionspec > [,...*n*] [WITH < termination >]

database

Is the name of the database changed.

ADD FILE

Specifies that a file is added.

TO FILEGROUP

Specifies the filegroup to which to add the specified file.

filegroup_name

Is the name of the filegroup to add the specified file to.

ADD LOG FILE

Specifies that a log file be added to the specified database.

REMOVE FILE

Removes the file description from the database system tables and deletes the physical file. The file cannot be removed unless empty.

ADD FILEGROUP

Specifies that a filegroup is to be added.

filegroup_name

Is the name of the filegroup to add or drop.

REMOVE FILEGROUP

Removes the filegroup from the database and deletes all the files in the filegroup. The filegroup cannot be removed unless empty.

MODIFY FILE

Specifies the given file that should be modified, including the FILENAME, SIZE, FILEGROWTH, and MAXSIZE options. Only one of these properties can be changed at a time. NAME must be specified in the <filespec> to identify the file to be modified. If SIZE is specified, the new size must be larger than the current file size. FILENAME can be specified only for files in the tempdb database, and the new name does not take effect until Microsoft SQL Server is restarted.

To modify the logical name of a data file or log file, specify in NAME the logical file name to be renamed, and specify for NEWNAME the new logical name for the file.

Thus:

MODIFY FILE (NAME = *logical_file_name*, NEWNAME = *new_logical_name*...).

For optimum performance during multiple modify-file operations, several ALTER DATABASE *database* MODIFY FILE statements can be run concurrently.

MODIFY NAME = *new_dbname*

Renames the database.

MODIFY FILEGROUP *filegroup_name* { *filegroup_property* | NAME = *new_filegroup_name* }

Specifies the filegroup to be modified and the change needed.

If *filegroup_name* and NAME = *new_filegroup_name* are specified, changes the filegroup name to the *new_filegroup_name*.

If *filegroup_name* and *filegroup_property* are specified, indicates the given filegroup property be applied to the filegroup. The values for *filegroup_property* are:

READONLY
Specifies the filegroup is read-only. Updates to objects in it are not allowed. The primary filegroup cannot be made read-only. Only users with exclusive database access can mark a filegroup read-only.

READWRITE
Reverses the READONLY property. Updates are enabled for the objects in the filegroup. Only users who have exclusive access to the database can mark a filegroup read/write.

DEFAULT
Specifies the filegroup as the default database filegroup. Only one database filegroup can be default. CREATE DATABASE sets the primary filegroup as the initial default filegroup. New tables and indexes are created in the default

filegroup—if no filegroup is specified in the CREATE TABLE, ALTER TABLE, or CREATE INDEX statements

WITH <termination>

Specifies when to roll back incomplete transactions when the database is transitioned from one state to another. Only one termination clause can be specified and it follows the SET clauses.

ROLLBACK AFTER *integer* [SECONDS] | ROLLBACK IMMEDIATE

Specifies whether to roll back after the specified number of seconds or immediately. If the termination clause is omitted, transactions are allowed to commit or roll back on their own.

NO_WAIT

Specifies that if the requested database state or option change cannot complete immediately without waiting for transactions to commit or roll back on their own, the request will fail.

<state_option>

Controls user access to the database, whether the database is online, and whether writes are allowed.

SINGLE_USER | RESTRICTED_USER | MULTI_USER

Controls which users may access the database. When SINGLE_USER is specified, only one user at a time can access the database. When RESTRICTED_USER is specified, only members of the db_owner, dbcreator, or sysadmin roles can use the database. MULTI_USER returns the database to its normal operating state.

OFFLINE | ONLINE

Controls whether the database is offline or online.

READ_ONLY | READ_WRITE

Specifies whether the database is in read-only mode. In read-only mode, users can read data from the database, not modify it. The database cannot be in use when READ_ONLY is specified. The master database is the

exception, and only the system administrator can use master while READ_ONLY is set. READ_WRITE returns the database to read/write operations.

<cursor_option>

Controls cursor options.

CURSOR_CLOSE_ON_COMMIT ON | OFF

If ON is specified, any cursors open when a transaction is committed or rolled back are closed. If OFF is specified, such cursors remain open when a transaction is committed; rolling back a transaction closes any cursors except those defined as INSENSITIVE or STATIC.

CURSOR_DEFAULTLOCAL | GLOBAL

Controls whether cursor scope defaults to LOCAL or GLOBAL.

<auto_option>

Controls automatic options.

AUTO_CLOSE ON | OFF

If ON is specified, the database is shut down cleanly and its resources are freed after the last user exits. If OFF is specified, the database remains open after the last user exits.

AUTO_CREATE_STATISTICS ON | OFF

If ON is specified, any missing statistics needed by a query for optimization are automatically built during optimization.

AUTO_SHRINK ON | OFF

If ON is specified, the database files are candidates for automatic periodic shrinking.

AUTO_UPDATE_STATISTICS ON | OFF

If ON is specified, any out-of-date statistics required by a query for optimization are automatically built during optimization. If OFF is specified, statistics must be updated manually.

<sql_option>

Controls the ANSI compliance options.

ANSI_NULL_DEFAULT ON | OFF
If ON is specified, CREATE TABLE follows SQL-92 rules to determine whether a column allows null values.
ANSI_NULLS ON | OFF
If ON is specified, all comparisons to a null value evaluate to UNKNOWN. If OFF is specified, comparisons of non-UNICODE values to a null value evaluate to TRUE if both values are NULL.
ANSI_PADDING ON | OFF
If ON is specified, strings are padded to the same length before comparison or insert. If OFF is specified, strings are not padded.
ANSI_WARNINGS ON | OFF
If ON is specified, errors or warnings are issued when conditions such as divide-by-zero occur.
ARITHABORT ON | OFF
If ON is specified, a query is terminated when an overflow or divide-by-zero error occurs during query execution.
CONCAT_NULL_YIELDS_NULL ON | OFF
If ON is specified, the result of a concatenation operation is NULL when either operand is NULL. If OFF is specified, the null value is treated as an empty character string. The default is OFF.
QUOTED_IDENTIFIER ON | OFF
If ON is specified, double quotation marks can be used to enclose delimited identifiers.
NUMERIC_ROUNDABORT ON | OFF
If ON is specified, an error is generated when loss of precision occurs in an expression.
RECURSIVE_TRIGGERS ON | OFF
If ON is specified, recursive firing of triggers is allowed. RECURSIVE_TRIGGERS OFF, the default, prevents direct recursion only. To disable indirect recursion as well, set the nested triggers server option to 0 using sp_configure.

<recovery_options>

Controls database recovery options.

RECOVERY FULL | BULK_LOGGED | SIMPLE

If FULL is specified, complete protection against media failure is provided. If a data file is damaged, media recovery can restore all committed transactions.

If BULK_LOGGED is specified, protection against media failure is combined with the best performance and least amount of log memory usage for certain large scale or bulk operations. These operations include SELECT INTO, bulk load operations (bcp and BULK INSERT), CREATE INDEX, and text and image operations (WRITETEXT and UPDATETEXT).

Under the bulk-logged recovery model, logging for the entire class is minimal and cannot be controlled on an operation-by-operation basis.

If SIMPLE is specified, a simple backup strategy that uses minimal log space is provided. Log space can be automatically reused when no longer needed for server failure recovery.

> **Important The simple recovery model is easier to manage than the other two models but at the expense of higher data loss exposure if a data file is damaged. All changes since the most recent database or differential database backup are lost and must be re-entered manually.**

The default recovery model is determined by the recovery model of the model database. To change the default for new databases, use ALTER DATABASE to set the recovery option of the model database.

TORN_PAGE_DETECTION ON | OFF

If ON is specified, incomplete pages can be detected. The default is ON.

Remarks

To remove a database, use DROP DATABASE. To rename a database, use sp_renamedb.Before you apply a different or new collation to a database, ensure the following conditions are in place:

1. **You are the only one currently using the database.**
2. **No schema bound object is dependent on the collation of the database.**

   If the following objects, which are dependent on the database collation, exist in the database, the `ALTER DATABASE database COLLATE` statement will fail. SQL Server will return an error message for each object blocking the ALTER action:

   - **User-defined functions and views created with SCHEMABINDING.**
   - **Computed columns.**
   - **CHECK constraints.**
   - **Table-valued functions that return tables with character columns with collations inherited from the default database collation.**

3. **Altering the database collation does not create duplicates among any system names for the database objects.**

   These namespaces may cause the failure of a database collation alteration if duplicate names result from the changed collation:

   - **Object names (such as procedure, table, trigger, or view).**
   - **Schema names (such as group, role, or user).**
   - **Scalar-type names (such as system and user-defined types).**

- Full-text catalog names.
- Column or parameter names within an object.
- Index names within a table.

Duplicate names resulting from the new collation will cause the alter action to fail and SQL Server will return an error message specifying the namespace where the duplicate was found.

You cannot add or remove a file while a BACKUP statement is executing.

To specify a fraction of a megabyte in the size parameters, convert the value to kilobytes by multiplying the number by 1024. For example, specify 1536 KB instead of 1.5MB (1.5 x 1024 = 1536).

Permissions

ALTER DATABASE permissions default to members of the sysadmin and dbcreator fixed server roles, and to members of the db_owner fixed database roles. These permissions are not transferable.

Examples

A. Add a file to a database

This example creates a database and alters it to add a new 5-MB data file.

```
USE master
GO
CREATE DATABASE Test1 ON
(
 NAME = Test1dat1,
 FILENAME = 'c:\Program Files\Microsoft SQL
Server\MSSQL\Data\t1dat1.ndf',
 SIZE = 5MB,
 MAXSIZE = 100MB,
```

```
 FILEGROWTH = 5MB
)
GO
ALTER DATABASE Test1
ADD FILE
(
 NAME = Test1dat2,
 FILENAME = 'c:\Program Files\Microsoft SQL
Server\MSSQL\Data\t1dat2.ndf',
 SIZE = 5MB,
 MAXSIZE = 100MB,
 FILEGROWTH = 5MB
)
GO
```

B. Add a filegroup with two files to a database

This example creates a filegroup in the Test 1 database created in Example A and adds two 5-MB files to the filegroup. It then makes Test1FG1 the default filegroup.

```
USE master
GO
ALTER DATABASE Test1
ADD FILEGROUP Test1FG1
GO

ALTER DATABASE Test1
ADD FILE
( NAME = test1dat3,
  FILENAME = 'c:\Program Files\Microsoft SQL
Server\MSSQL\Data\t1dat3.ndf',
  SIZE = 5MB,
  MAXSIZE = 100MB,
  FILEGROWTH = 5MB),
( NAME = test1dat4,
  FILENAME = 'c:\Program Files\Microsoft SQL
Server\MSSQL\Data\t1dat4.ndf',
  SIZE = 5MB,
  MAXSIZE = 100MB,
  FILEGROWTH = 5MB)
```

```
TO FILEGROUP Test1FG1

ALTER DATABASE Test1
MODIFY FILEGROUP Test1FG1 DEFAULT
GO
```

C. Add two log files to a database

This example adds two 5-MB log files to a database.

```
USE master
GO
ALTER DATABASE Test1
ADD LOG FILE
( NAME = test1log2,
  FILENAME = 'c:\Program Files\Microsoft SQL
Server\MSSQL\Data\test2log.ldf',
  SIZE = 5MB,
  MAXSIZE = 100MB,
  FILEGROWTH = 5MB),
( NAME = test1log3,
  FILENAME = 'c:\Program Files\Microsoft SQL
Server\MSSQL\Data\test3log.ldf',
  SIZE = 5MB,
  MAXSIZE = 100MB,
  FILEGROWTH = 5MB)
GO
```

D. Remove a file from a database

This example removes one of the files added to the Test1 database in Example B.

```
USE master
GO
ALTER DATABASE Test1
REMOVE FILE test1dat4
GO
```

E. Modify a file

This example increases the size of one of the files added to the Test1 database in Example B.

```
USE master
GO
ALTER DATABASE Test1
MODIFY FILE
    (NAME = test1dat3,
    SIZE = 20MB)
GO
```

F. Make the primary filegroup the default

This example makes the primary filegroup the default filegroup if another filegroup was made the default earlier.

```
USE master
GO
ALTER DATABASE MyDatabase
MODIFY FILEGROUP [PRIMARY] DEFAULT
GO
```

3-Sql-Server Data Types:

Exact Numerics

Integers

bigint

Integer (whole number) data from -2^63 (-9223372036854775808) through 2^63-1 (9223372036854775807).

int

Integer (whole number) data from -2^31 (-2,147,483,648) through 2^31 - 1 (2,147,483,647).

smallint

Integer data from 2^15 (-32,768) through 2^15 - 1 (32,767).

tinyint

Integer data from 0 through 255.

bit

bit

Integer data with either a 1 or 0 value.

decimal and numeric

decimal

Fixed precision and scale numeric data from -10^38 +1 through 10^38 –1.

numeric

Functionally equivalent to decimal.

money and smallmoney

money

Monetary data values from -2^63 (-922,337,203,685,477.5808) through 2^63 - 1 (+922,337,203,685,477.5807), with accuracy to a ten-thousandth of a monetary unit.

smallmoney

Monetary data values from -214,748.3648 through +214,748.3647, with accuracy to a ten-thousandth of a monetary unit.

Approximate Numerics

float

Floating precision number data from -1.79E + 308 through 1.79E + 308.

real

Floating precision number data from -3.40E + 38 through 3.40E + 38.

datetime and smalldatetime

datetime

Date and time data from January 1, 1753, through December 31, 9999, with an accuracy of three-hundredths of a second, or 3.33 milliseconds.

smalldatetime

Date and time data from January 1, 1900, through June 6, 2079, with an accuracy of one minute.

Character Strings

char

Fixed-length non-Unicode character data with a maximum length of 8,000 characters.

varchar

Variable-length non-Unicode data with a maximum of 8,000 characters.

text

Variable-length non-Unicode data with a maximum length of 2^31 - 1 (2,147,483,647) characters.

Unicode Character Strings

nchar

Fixed-length Unicode data with a maximum length of 4,000 characters.

nvarchar

Variable-length Unicode data with a maximum length of 4,000 characters. sysname is a system-supplied user-defined data type that is functionally equivalent to nvarchar(128) and is used to reference database object names.

ntext

Variable-length Unicode data with a maximum length of 2^30 - 1 (1,073,741,823) characters.

Binary Strings

binary

Fixed-length binary data with a maximum length of 8,000 bytes.

varbinary

Variable-length binary data with a maximum length of 8,000 bytes.

image

Variable-length binary data with a maximum length of 2^31 - 1 (2,147,483,647) bytes.

4-CREATE TABLE

Creates a new table.

Syntax

```
CREATE TABLE
   [ database_name.[ owner ] . | owner. ] table_name
   ( { < column_definition >
      | column_name AS computed_column_expression
      | < table_constraint > ::= [ CONSTRAINT constraint_name ] }

         | [ { PRIMARY KEY | UNIQUE } [ ,...n ]
   )

[ ON { filegroup | DEFAULT } ]

< column_definition > ::= { column_name data_type }
   [ [ DEFAULT constant_expression ]
      | [ IDENTITY [ ( seed , increment ) [ NOT FOR REPLICATION
] ] ]
   ]

< column_constraint > ::= [ CONSTRAINT constraint_name ]
   { [ NULL | NOT NULL ]
      | [ { PRIMARY KEY | UNIQUE }
         [ CLUSTERED | NONCLUSTERED ]
         [ON {filegroup | DEFAULT} ] ]
      ]
      | [ [ FOREIGN KEY ]
         REFERENCES ref_table [ ( ref_column ) ]
      ]
      | CHECK [ NOT FOR REPLICATION ]
      ( logical_expression )
   }

< table_constraint > ::= [ CONSTRAINT constraint_name ]
   { [ { PRIMARY KEY | UNIQUE }
      [ CLUSTERED | NONCLUSTERED ]
      { ( column [ ASC | DESC ] [ ,...n ] ) }
      [ ON { filegroup | DEFAULT } ]
   ]
   | FOREIGN KEY
      [ ( column [ ,...n ] ) ]
```

REFERENCES *ref_table* [(*ref_column* [,...*n*])]
(*search_conditions*)
}

Arguments

table_name

Is the name of the new table. Table names must conform to the rules for identifiers. The combination of *owner.table_name* must be unique within the database. *table_name* can contain a maximum of 128 characters, except for local temporary table names (names prefixed with a single number sign (#)) that cannot exceed 116 characters.

column_name

Is the name of a column in the table. Column names must conform to the rules for identifiers and must be unique in the table..

computed_column_expression

A computed column cannot be used as a DEFAULT or FOREIGN KEY constraint definition or with a NOT NULL constraint definition.

- **A computed column cannot be the target of an INSERT or UPDATE statement.**

  The nullability of computed columns is determined automatically by SQL Server based on the expressions used.

ON {*filegroup* | DEFAULT}

Specifies the filegroup on which the table is stored.

ON {*filegroup* | DEFAULT} can also be specified in a PRIMARY KEY or UNIQUE constraint. These constraints create indexes. If *filegroup* is specified, the index is stored in the named filegroup.

If DEFAULT is specified, the index is stored in the default filegroup. If no filegroup is specified in a constraint, the index is stored on the same filegroup as the table.

DEFAULT

Specifies the value provided for the column when a value is not explicitly supplied during an insert.

constant_expression

Is a constant, NULL, or a system function used as the default value for the column.

IDENTITY

Indicates that the new column is an identity column. When a new row is added to the table, Microsoft® SQL Server™ provides a unique, incremental value for the column. Identity columns are commonly used in conjunction with PRIMARY KEY constraints to serve as the unique row identifier for the table. The IDENTITY property can be assigned to tinyint, smallint, int, bigint, decimal(p,0), or numeric(p,0) columns. Only one identity column can be created per table. Bound defaults and DEFAULT constraints cannot be used with an identity column. You must specify both the seed and increment or neither. If neither is specified, the default is (1,1).

seed

Is the value used for the very first row loaded into the table.

increment

Is the incremental value added to the identity value of the previous row loaded.

CONSTRAINT

Is an optional keyword indicating the beginning of a PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY, or CHECK constraint definition. Constraints are special properties that enforce data integrity and they may create indexes for the table and its columns.

constraint_name

Is the name of a constraint. Constraint names must be unique within a database.

NULL | NOT NULL

Are keywords that determine if null values are allowed in the column. NULL is not strictly a constraint but can be specified in the same manner as NOT NULL.

PRIMARY KEY

Is a constraint that enforces entity integrity for a given column or columns through a unique index. Only one PRIMARY KEY constraint can be created per table.

UNIQUE

Is a constraint that provides entity integrity for a given column or columns through a unique index. A table can have multiple UNIQUE constraints.

CLUSTERED | NONCLUSTERED

Are keywords to indicate that a clustered or a nonclustered index is created for the PRIMARY KEY or UNIQUE constraint. PRIMARY KEY constraints default to CLUSTERED and UNIQUE constraints default to NONCLUSTERED.

You can specify CLUSTERED for only one constraint in a CREATE TABLE statement. If you specify CLUSTERED for a UNIQUE constraint and also specify a PRIMARY KEY constraint, the PRIMARY KEY defaults to NONCLUSTERED.

FOREIGN KEY...REFERENCES

Is a constraint that provides referential integrity for the data in the column or columns. FOREIGN KEY constraints require that each value in the column exists in the corresponding referenced column(s) in the referenced table. FOREIGN KEY constraints can reference only columns that are PRIMARY KEY or UNIQUE constraints in the referenced table or columns referenced in a UNIQUE INDEX on the referenced table.

ref_table

Is the name of the table referenced by the FOREIGN KEY constraint.

(*ref_column*[,...*n*])

Is a column, or list of columns, from the table referenced by the FOREIGN KEY constraint.

CHECK

Is a constraint that enforces domain integrity by limiting the possible values that can be entered into a column or columns.

logical_expression

Is a logical expression that returns TRUE or FALSE.

column

Is a column or list of columns, in parentheses, used in table constraints to indicate the columns used in the constraint definition.

[ASC | DESC]

Specifies the order in which the column or columns participating in table constraints are sorted. The default is ASC.

n

Is a placeholder indicating that the preceding item can be repeated *n* number of times.

Examples

A. Use PRIMARY KEY constraints

This example shows the column definition for a PRIMARY KEY constraint with a clustered index on the job_id column of the jobs table (allowing the system to supply the constraint name) in the pubs sample database.

```
job_id   smallint
      PRIMARY KEY CLUSTERED
```

This example shows how a name can be supplied for the PRIMARY KEY constraint. This constraint is used on the emp_id column of the employee table. This column is based on a user-defined data type.

```
emp_id   empid
      CONSTRAINT PK_emp_id PRIMARY KEY NONCLUSTERED
```

B. Use FOREIGN KEY constraints

A FOREIGN KEY constraint is used to reference another table. Foreign keys can be single-column keys or multicolumn keys. This example shows a single-column FOREIGN KEY constraint on the employee table that references the jobs table. Only the REFERENCES clause is required for a single-column FOREIGN KEY constraint.

```
job_id   smallint      NOT NULL
      DEFAULT 1
      REFERENCES jobs(job_id)
```

You can also explicitly use the FOREIGN KEY clause and restate the column attribute. Note that the column name does not have to be the same in both tables.

```
FOREIGN KEY (job_id) REFERENCES jobs(job_id)
```

Multicolumn key constraints are created as table constraints. In the pubs database, the sales table includes a multicolumn PRIMARY KEY. This example shows how to reference this key from another table; an explicit constraint name is optional.

```
CONSTRAINT FK_sales_backorder FOREIGN KEY (stor_id,
ord_num, title_id)
   REFERENCES sales (stor_id, ord_num, title_id)
```

C. Use UNIQUE constraints

UNIQUE constraints are used to enforce uniqueness on nonprimary key columns. A PRIMARY KEY constraint column includes a restriction for uniqueness automatically; however, a UNIQUE constraint can allow null values. This example shows a column called pseudonym on the authors table. It enforces a restriction that authors' pen names must be unique.

```
pseudonym varchar(30)    NULL
UNIQUE NONCLUSTERED
```

This example shows a UNIQUE constraint created on the stor_name and city columns of the stores table, where the stor_id is actually the PRIMARY KEY; no two stores in the same city should be the same.

```
CONSTRAINT U_store UNIQUE NONCLUSTERED (stor_name,
city)
```

D. Use DEFAULT definitions

Defaults supply a value (with the INSERT and UPDATE statements) when no value is supplied. In the pubs database, many DEFAULT definitions are used to ensure that valid data or placeholders are entered.

On the jobs table, a character string default supplies a description (column job_desc) when the actual description is not entered explicitly.

```
DEFAULT 'New Position - title not formalized yet'
```

In the employee table, the employees can be employed by an imprint company or by the parent company. When an explicit company is not supplied, the parent company is entered (note that, as shown here, comments can be nested within the table definition).

```
DEFAULT ('9952')
/* By default the Parent Company Publisher is the
company
to whom each employee reports. */
```

In addition to constants, DEFAULT definitions can include functions. Use this example to get the current date for an entry:

```
DEFAULT (getdate())
```

Niladic-functions can also improve data integrity. To keep track of the user who inserted a row, use the niladic-function for USER (do not surround the niladic-functions with parentheses):

```
DEFAULT USER
```

E. Use CHECK constraints

This example shows restrictions made to the values entered into the min_lvl and max_lvl columns of the jobs table. Both of these constraints are unnamed:

```
CHECK (min_lvl >= 10)
```

and

```
CHECK (max_lvl <= 250)
```

This example shows a named constraint with a pattern restriction on the character data entered into the emp_id column of the employee table.

```
CONSTRAINT CK_emp_id CHECK (emp_id LIKE
   '[A-Z][A-Z][A-Z][1-9][0-9][0-9][0-9][0-9][FM]'
OR
   emp_id LIKE '[A-Z]-[A-Z][1-9][0-9][0-9][0-9][0-
9][FM]')
```

This example specifies that the pub_id must be within a specific list or follow a given pattern. This constraint is for the pub_id of the publishers table.

```
CHECK (pub_id IN ('1389', '0736', '0877', '1622',
'1756')
   OR pub_id LIKE '99[0-9][0-9]')
```

F. Complete table definitions

This example shows complete table definitions with all constraint definitions for three tables (jobs, employee, and publishers) created in the pubs database.

```
/* ************************** jobs table
************************** */
CREATE TABLE jobs
(
   job_id  smallint
      IDENTITY(1,1)
      PRIMARY KEY CLUSTERED,
   job_desc        varchar(50)     NOT NULL
      DEFAULT 'New Position - title not formalized
yet',
   min_lvl tinyint NOT NULL
      CHECK (min_lvl >= 10),
   max_lvl tinyint NOT NULL
      CHECK (max_lvl <= 250)
)
```

```
/* ************************ employee table
************************ */
CREATE TABLE employee
(
   emp_id  empid
      CONSTRAINT PK_emp_id PRIMARY KEY NONCLUSTERED
      CONSTRAINT CK_emp_id CHECK (emp_id LIKE
         '[A-Z][A-Z][A-Z][1-9][0-9][0-9][0-9][0-
9][FM]' or
         emp_id LIKE '[A-Z]-[A-Z][1-9][0-9][0-9][0-
9][0-9][FM]'),
      /* Each employee ID consists of three
characters that
      represent the employee's initials, followed
by a five
      digit number ranging from 10000 through 99999
and then the
      employee's gender (M or F). A (hyphen) - is
acceptable
      for the middle initial. */
   fname   varchar(20)     NOT NULL,
   minit   char(1) NULL,
   lname   varchar(30)     NOT NULL,
   job_id  smallint        NOT NULL
      DEFAULT 1
      /* Entry job_id for new hires. */
      REFERENCES jobs(job_id),
   job_lvl tinyint
      DEFAULT 10,
      /* Entry job_lvl for new hires. */
   pub_id  char(4) NOT NULL
      DEFAULT ('9952')
      REFERENCES publishers(pub_id),
      /* By default, the Parent Company Publisher
is the company
      to whom each employee reports. */
   hire_date       datetime        NOT NULL
      DEFAULT (getdate())
      /* By default, the current system date is
entered. */
```

```
)

/* ***************** publishers table
******************** */
CREATE TABLE publishers
(
   pub_id  char(4) NOT NULL
         CONSTRAINT UPKCL_pubind PRIMARY KEY
CLUSTERED
         CHECK (pub_id IN ('1389', '0736', '0877',
'1622', '1756')
            OR pub_id LIKE '99[0-9][0-9]'),
   pub_name      varchar(40)     NULL,
   city          varchar(20)     NULL,
   state      char(2) NULL,
   country       varchar(30)     NULL
            DEFAULT('USA')
)
```

G. Use the uniqueidentifier data type in a column

This example creates a table with a uniqueidentifier column. It uses a PRIMARY KEY constraint to protect the table against users inserting duplicated values, and it uses the NEWID() function in the DEFAULT constraint to provide values for new rows.

```
CREATE TABLE Globally_Unique_Data
(guid uniqueidentifier
   CONSTRAINT Guid_Default
   DEFAULT NEWID(),
Employee_Name varchar(60),
CONSTRAINT Guid_PK PRIMARY KEY (Guid)
)
```

H. Use an expression for a computed column

This example illustrates the use of an expression ((low + high)/2) for calculating the myavg computed column.

```
CREATE TABLE mytable
```

```
    (
     low int,
     high int,
     myavg AS (low + high)/2
    )
```

I. Use the USER_NAME function for a computed column

This example uses the USER_NAME function in the myuser_name column.

```
CREATE TABLE mylogintable
    (
     date_in datetime,
     user_id int,
     myuser_name AS USER_NAME()
    )
```

5-ALTER TABLE

Modifies a table definition by altering, adding, or dropping columns and constraints, or by disabling or enabling constraints and triggers.

Syntax

ALTER TABLE *table*
{ [ALTER COLUMN *column_name*
{ *new_data_type* [(*precision* [, *scale*])]
[NULL | NOT NULL]
]
| ADD
{ [< column_definition >]
| *column_name* AS *computed_column_expression*
} [,...*n*]
| [WITH CHECK | WITH NOCHECK] ADD
{ < table_constraint > } [,...*n*]
| DROP
{ [CONSTRAINT] *constraint_name*
| COLUMN *column* } [,...*n*]
| { CHECK | NOCHECK } CONSTRAINT

```
        { ALL | constraint_name [ ,...n ] }
    | { ENABLE | DISABLE } TRIGGER
        { ALL | trigger_name [ ,...n ] }
}

< column_definition > ::=
    { column_name data_type }
    [ [ DEFAULT constant_expression ] [ WITH VALUES ]
    | [ IDENTITY [ (seed , increment ) [ NOT FOR REPLICATION ] ] ]
        ]
    [ < column_constraint > ] [ ...n ]

< column_constraint > ::=
    [ CONSTRAINT constraint_name ]
    { [ NULL | NOT NULL ]
        | [ { PRIMARY KEY | UNIQUE }
            [ CLUSTERED | NONCLUSTERED ]
            [ ON { filegroup | DEFAULT } ]
            ]
        | [ [ FOREIGN KEY ]
            REFERENCES ref_table [ ( ref_column ) ]
            ]
            ( logical_expression )
    }

< table_constraint > ::=
    [ CONSTRAINT constraint_name ]
    { [ { PRIMARY KEY | UNIQUE }
        [ CLUSTERED | NONCLUSTERED ]
        { ( column [ ,...n ] ) }
        [ ON {filegroup | DEFAULT } ]
        ]
        |   FOREIGN KEY
            [ ( column [ ,...n ] ) ]
            REFERENCES ref_table [ ( ref_column [ ,...n ] ) ]
        | DEFAULT constant_expression
            [ FOR column ] [ WITH VALUES ]
        |   CHECK [ NOT FOR REPLICATION ]
            ( search_conditions )
    }
```

Arguments

ALTER COLUMN

Specifies that the given column is to be changed or altered. The altered column cannot be:

- **A column with a text, image, ntext, or timestamp data type.**
- **A computed column or used in a computed column.**
- **Used in an index, unless the column is a varchar, nvarchar, or varbinary data type, the data type is not changed, and the new size is equal to or larger than the old size.**
- **Used in a PRIMARY KEY or [FOREIGN KEY] REFERENCES constraint.**
- **Used in a CHECK or UNIQUE constraint, except that altering the length of a variable-length column used in a CHECK or UNIQUE constraint is allowed.**
- **Associated with a default, except that changing the length, precision, or scale of a column is allowed if the data type is not changed.**

new_data_type

Is the new data type for the altered column. Criteria for the *new_data_type* of an altered column are:

- **The previous data type must be implicitly convertible to the new data type.**
- **If the altered column is an identity column, *new_data_type* must be a data type that supports the identity property.**

precision

Is the precision for the specified data type.

scale

Is the scale for the specified data type.

NULL | NOT NULL

Specifies whether the column can accept null values. Columns that do not allow null values can be added with ALTER TABLE only if they have a default specified. A new column added to a table must either allow null values, or the column must be specified with a default value.

If the new column allows null values and no default is specified, the new column contains a null value for each row in the table. If the new column allows null values and a default definition is added with the new column, the WITH VALUES option can be used to store the default value in the new column for each existing row in the table.

If the new column does not allow null values, a DEFAULT definition must be added with the new column, and the new column automatically loads with the default value in the new columns in each existing row.

NULL can be specified in ALTER COLUMN to make a NOT NULL column allow null values, except for columns in PRIMARY KEY constraints. NOT NULL can be specified in ALTER COLUMN only if the column contains no null values. The null values must be updated to some value before the ALTER COLUMN NOT NULL is allowed, such as:

```
UPDATE MyTable SET NullCol = N'some_value' WHERE
NullCol IS NULL

ALTER TABLE MyTable ALTER COLUMN NullCOl
NVARCHAR(20) NOT NULL
```

If NULL or NOT NULL is specified with ALTER COLUMN, *new_data_type* [(*precision* [, *scale*])] must also be specified. If the data type, precision, and scale are not changed, specify the current column values.

ADD

Specifies that one or more column definitions, computed column definitions, or table constraints are added.

DROP { [CONSTRAINT] *constraint_name* | COLUMN *column_name* }

Specifies that *constraint_name* or *column_name* is removed from the table. DROP COLUMN is not allowed if the compatibility level is 65 or earlier. Multiple columns and constraints can be listed. A column cannot be dropped if it is:

- **Used in an index.**
- **Used in a CHECK, FOREIGN KEY, UNIQUE, or PRIMARY KEY constraint.**
- **Associated with a default defined with the DEFAULT keyword, or bound to a default object.**

{ CHECK | NOCHECK} CONSTRAINT

Specifies that *constraint_name* is enabled or disabled. When disabled, future inserts or updates to the column are not validated against the constraint conditions. This option can only be used with FOREIGN KEY and CHECK constraints.

ALL

Specifies that all constraints are disabled with the NOCHECK option, or enabled with the CHECK option.

{ENABLE | DISABLE} TRIGGER

Specifies that *trigger_name* is enabled or disabled. When a trigger is disabled it is still defined for the table; however, when INSERT, UPDATE, or DELETE statements are executed against the table, the actions in the trigger are not performed until the trigger is re-enabled.

ALL

Specifies that all triggers in the table are enabled or disabled.

trigger_name

Specifies the name of the trigger to disable or enable.

column_name data_type

Is the data type for the new column. *data_type* can be any Microsoft® SQL Server™ or user-defined data type.

DEFAULT

Is a keyword that specifies the default value for the column. DEFAULT definitions can be used to provide values for a new column in the existing rows of data. DEFAULT definitions cannot be added to columns that have a timestamp data type, an IDENTITY property, an existing DEFAULT definition, or a bound default. If the column has an existing default, the default must be dropped before the new default can be added. To maintain compatibility with earlier versions of SQL Server, it is possible to assign a constraint name to a DEFAULT.

Permissions

ALTER TABLE permissions default to the table owner, members of the sysadmin fixed server role, and the db_owner and db_ddladmin fixed database roles, and are not transferable.

Examples

A. Alter a table to add a new column

This example adds a column that allows null values and has no values provided through a DEFAULT definition. Each row will have a NULL in the new column.

```
CREATE TABLE doc_exa ( column_a INT)
GO
ALTER TABLE doc_exa ADD column_b VARCHAR(20) NULL
```

```
GO
EXEC sp_help doc_exa
GO
DROP TABLE doc_exa
GO
```

B. Alter a table to drop a column

This example modifies a table to remove a column.

```
CREATE TABLE doc_exb ( column_a INT, column_b
VARCHAR(20) NULL)
GO
ALTER TABLE doc_exb DROP COLUMN column_b
GO
EXEC sp_help doc_exb
GO
DROP TABLE doc_exb
GO
```

C. Alter a table to add a column with a constraint

This example adds a new column with a UNIQUE constraint.

```
CREATE TABLE doc_exc ( column_a INT)
GO
ALTER TABLE doc_exc ADD column_b VARCHAR(20) NULL
   CONSTRAINT exb_unique UNIQUE
GO
EXEC sp_help doc_exc
GO
DROP TABLE doc_exc
GO
```

D. Alter a table to add an unverified constraint

This example adds a constraint to an existing column in the table. The column has a value that violates the constraint; therefore, WITH NOCHECK is used to prevent the constraint from being validated against existing rows, and to allow the constraint to be added.

```
CREATE TABLE doc_exd ( column_a INT)
GO
INSERT INTO doc_exd VALUES (-1)
GO
ALTER TABLE doc_exd WITH NOCHECK
ADD CONSTRAINT exd_check CHECK (column_a > 1)
GO
EXEC sp_help doc_exd
GO
DROP TABLE doc_exd
GO
```

E. Alter a table to add several columns with constraints

This example adds several columns with constraints defined with the new column. The first new column has an IDENTITY property; each row in the table has new incremental values in the identity column.

```
CREATE TABLE doc_exe ( column_a INT CONSTRAINT
column_a_un UNIQUE)
GO
ALTER TABLE doc_exe ADD

/* Add a PRIMARY KEY identity column. */
column_b INT IDENTITY
CONSTRAINT column_b_pk PRIMARY KEY,

/* Add a column referencing another column in the
same table. */
column_c INT NULL
CONSTRAINT column_c_fk
REFERENCES doc_exe(column_a),

/* Add a column with a constraint to enforce that
*/
/* nonnull data is in a valid phone number format.
*/
column_d VARCHAR(16) NULL
CONSTRAINT column_d_chk
CHECK
```

```
(column_d IS NULL OR
column_d LIKE "[0-9][0-9][0-9]-[0-9][0-9][0-9][0-
9]" OR
column_d LIKE
"([0-9][0-9][0-9]) [0-9][0-9][0-9]-[0-9][0-9][0-
9][0-9]"),

/* Add a nonnull column with a default.  */
column_e DECIMAL(3,3)
CONSTRAINT column_e_default
DEFAULT .081
GO
EXEC sp_help doc_exe
GO
DROP TABLE doc_exe
GO
```

F. Add a nullable column with default values

This example adds a nullable column with a DEFAULT definition, and uses WITH VALUES to provide values for each existing row in the table. If WITH VALUES is not used, each row has the value NULL in the new column.

```
ALTER TABLE MyTable
ADD AddDate smalldatetime NULL
CONSTRAINT AddDateDflt
DEFAULT getdate() WITH VALUES
```

G. Disable and reenable a constraint

This example disables a constraint that limits the salaries accepted in the data. WITH NOCHECK CONSTRAINT is used with ALTER TABLE to disable the constraint and allow an insert that would normally violate the constraint. WITH CHECK CONSTRAINT re-enables the constraint.

```
CREATE TABLE cnst_example
(id INT NOT NULL,
 name VARCHAR(10) NOT NULL,
 salary MONEY NOT NULL
```

```
    CONSTRAINT salary_cap CHECK (salary < 100000)
)

-- Valid inserts
INSERT INTO cnst_example VALUES (1,"Joe
Brown",65000)
INSERT INTO cnst_example VALUES (2,"Mary
Smith",75000)

-- This insert violates the constraint.
INSERT INTO cnst_example VALUES (3,"Pat
Jones",105000)

-- Disable the constraint and try again.
ALTER TABLE cnst_example NOCHECK CONSTRAINT
salary_cap
INSERT INTO cnst_example VALUES (3,"Pat
Jones",105000)

-- Reenable the constraint and try another insert,
will fail.
ALTER TABLE cnst_example CHECK CONSTRAINT
salary_cap
INSERT INTO cnst_example VALUES (4,"Eric
James",110000)
```

H. Disable and reenable a trigger

This example uses the DISABLE TRIGGER option of ALTER TABLE to disable the trigger and allow an insert that would normally violate the trigger. It then uses ENABLE TRIGGER to re-enable the trigger.

```
CREATE TABLE trig_example
(id INT,
name VARCHAR(10),
salary MONEY)
go
-- Create the trigger.
CREATE TRIGGER trig1 ON trig_example FOR INSERT
as
```

```
IF (SELECT COUNT(*) FROM INSERTED
WHERE salary > 100000) > 0
BEGIN
print "TRIG1 Error: you attempted to insert a
salary > $100,000"
ROLLBACK TRANSACTION
END
GO
-- Attempt an insert that violates the trigger.
INSERT INTO trig_example VALUES (1,"Pat
Smith",100001)
GO
-- Disable the trigger.
ALTER TABLE trig_example DISABLE TRIGGER trig1
GO
-- Attempt an insert that would normally violate
the trigger
INSERT INTO trig_example VALUES (2,"Chuck
Jones",100001)
GO
-- Re-enable the trigger.
ALTER TABLE trig_example ENABLE TRIGGER trig1
GO
-- Attempt an insert that violates the trigger.
INSERT INTO trig_example VALUES (3,"Mary
Booth",100001)
GO
```

آزمایش 2

توابع در **SQL-SERVER**

پیش آگاهی

مقدمه :مروری بر توابع ورویه ها

همان گونه که درزبان های برنامه نویسی، امکان تعریف تابع ورویه وجود دارد،درمحیط **Sql-server**نیزاستفاده از آنها امکانپذیراست. توابع ورویه ها جز اشیاء هر پایگاه داده هستند و در دیکشنری داده ها نگهداری می شوند. درابتدای ایجادیک تابع یاروال ،ساختاردستورات آن هاکنترل شده ،خطایابی انجام می شود ، سپس نام تابع یارویه ایجادشده درجدول **sysobjects** ومتن آنهادرجدول**syscomments** ذخیره شده ، دراولین اجراکامپایل می شوند. بنابراین درفراخوانی های بعدی، درصورتی که نیازبه کامپایل دوباره آنهاازسوی کاربردرخواست نشود،کامپایل نمی شوند. این دوساختار کاربرد های دیگری نیز دارند از جمله:

1- کاهش بار شبکه : به جای فرستادن متن کامل با یک دستور تقا ضای اجرا ی یک تابع یـا روال را می توان اعلان کرد.

2- جایگزین های مناسب دیدها : می توان از روال ها و مخصوصـا توابـع بـه عنـوان جـایگزین هـای مناسبی برای دیدها استفاده کرد. یکی از بهترین دلایل استفاده از توابع به جای دیدها این اسـت که توابع مانند دیدها به یک دستور **select** محدود نیستند و می تواننـد هـر تعـداد دسـتور را اجرا کنند. مجوزهای دسترسی که در دیدها مطرح می شـوند در توابـع هـم قابـل پیـاده سـازی هستند .

توابع مورد استفاده در **SQL** به دو بخش تقسیم می شوند :

1- توابع سیستمی :

- توابع رشته ای
- توابع تاریخ
- توابع ریاضی
- سایرتوابع

2 - توابعی که توسط کاربر تعریف می شوند (**UDF**).

- توابع سیستمی

توابع رشته اي :

تابع	شرح
‘expression’ + ‘expression’	دو یا چند رشته را به هم می چسباند
PATINDEX(‘%pattern%’, expression)	محل نخستین وقوع **pattern** را برمی گرداند.
LTRIM(char_expr)	.حروف **blank** سمت چپ رشته را حذف می کند .

هنگام کار با اطلاعات کاراکتری ، توابع رشته ای زیادی برای پردازش وجود دارد . اکثر توابع رشته ای بر داده های نوع **nvarchar،varchar،nchar،char**کار می کند .

برخی از توابع رشته ای که کاربرد بیشتری دارند درجدول صفحه بعد ذکر شده اند :

RTRIM(char_expr)	حروف **blank** سمت راست رشته را حذف می کند .
REPLACE(‘string1’,’string2’,’string3’)	رشته**string1** در**string2**را با**string3**جایگزین می کند.
STUFF (expression1 , start , length , expression2)	در رشته **expression1** از محل **start** رشته ای به طول **length** را حذف نموده و **expression2** رادرمحل **start**جایگزین می کند.
SUBSTRING (expression , start , length)	زیررشته ای از **expression**باطول**length**وبا شروع از محل **start**را برمی گرداند.
UNICODE('ncharacter_expression')	مقدارصحیح **Unicode** نخستین کاراکتر رشته را بر می گرداند.
LEN (string_expression)	طول رشته را برمی گرداند.

مثال1 : لیست دروس دانشکده ای با کد **11** شامل نام ونوع درس:

```
Select Cname , desc
                                From  CRS,CODEFILE
        Where  (Substring(CRS.c#,1,2) =  ‘11’  AND
CODEFILE.field=’crstype’
                    And  CODEFILE.Type = CRS.CrsType)
```

مثال 2 : نام و نام خانوادگی و شماره ترم دانشجویانی که معدل کل آن ها در سال 83 بیشتر از 17 می باشد :

```
Select  Name , Family , 'Term'+Right(TrmNo,1)
    From STD , STDTRM
        Where
( STD.S# = STDTRM.S#    and   STD.Gpa >= 17   and
      Substring (STDTRM.TrmNo, 1 , 2 ) = 83  )
```

توابع تاریخ

این توابع برای پردازش برروی نوع داده ای **datetime** به کارمی رود :

SELECT date_function (parameters)

تابع	شرح
DATEPART (datepart , date)	مقدار عددی **datepart**ر ا در تاریخ برمی گرداند .**(datepart** می تواند یکی از اجزاء **date** ، یعنی سال(**yy)** ، ماه **(mm)** یا روز**(dd)** باشد.**)**
DATEDIFF(depart , date1 , date2)	تعداد **dateparte**های بین دو تاریخ را برمی گرداند.
GETDATE()	تاریخ و ساعت جاری را برمی گرداند .
DAY(date)	مقدار عددی نشان دهنده روز را بر میگرداند .
MONTH(date)	مقدار عددی نشان دهنده ماه بر میگرداند .
YEAR(date)	مقدار عددی نشان دهنده سال را بر میگرداند .

مثال3 :شماره دانشجویی وسن دانشجویانی که سال تولد آنها بزرگ تر از سال 1986 است:

```
SELECT  s#,datediff (yy,getdate( ),birthdate)
 FROM  STD    WHERE
     Datepart(yy,birthdate)>1986
```

توابع تعریف شده توسط کاربر(User Defined Function)

یکی از ویژگی های **SQL- Server** امکان تعریف توابع جدیدتوسط کاربر (**UDF)** می باشدکه با استفاده از این امکان ، می توان عملیات خاص مورد نیاز هربرنامه کاربردی را فقط یک باربه صورت یک تابع نوشت ودرموارد لازم،آن تابع را فراخواند تا عملیات مورد نظر انجام شود.

در ساختار آن می توان از توابع سیستمی ،توابع تعریف شده توسط کاربر، همچنین دستورات **T-SQL**و رویه های ذخیره شده استفاده کرد.

قبل از بررسی نحوه تعریف و به کار گیری این توابع به توضیح بعضی از ساختارهای دستوری پرکاربرد **T-sql** می پردازیم .

تعریف متغیرها

مانند اکثر زبان ها ی برنامه نویسی در **T-sql**نیز متغیر ها بعد از تعریف و تعیین نوع قابل استفاده می باشند .

متغیر ها در دو نوع قابل دسته بندی هستند :

متغیرهای عمومی : این متغیر ها با @@شروع می شوند .تمامی این متغیر ها توسط **Sql-server**تعریف و مقداردهی می شوند. مثلا**@@VERSION** که نسخه ونوع پردازنده مورد استفاده سرور رانمایش می دهد.از دیگر متغیر های مهم این گروه **@@IDENTITY**و**@@ROWCOUNT**هستند که به ترتیب آخرین مقدار**IDENTITY** اختصاص داده شده توسط سروروتعداد سطرهایی که تحت تاثیر آخرین دستور قرا ر گرفته اند را نمایش می دهند .

1- متغیر های محلی:این متغیر ها با @ شروع می شوند .

نحوه تعریف :

DECLARE @VARNAME DATA_TYPE|TABLE ({TABLE-DEFINITION})

دقت کنید که نوع داده می تواند **TABLE** هم باشد.

مثال**4** :

DECLARE @citytable (cityname CHAR(16),citycode int)

نحوه مقداردهی: برای مقدار دهی یک متغیر محلی روشهای مختلفی وجود دارد از جمله:

مثال **5** : استفاده از **SET**برای مقدار دهی : **@cityName='Isfahan'**

SET

مثال**6** : استفاده از **SELECT**برای مقدار دهی:

SELECT @myGpa=Gpa

From STD where s#='4351'

یا **SELECT @myTable=Name,Family FROM STD**

متغیر**@myTable**راکه قبلا از نوع جدول تعریف شده را با نام ونام خانوادگی دانشجویان جدول **STD**پر می کند .

دستورات اجرایی

برای کنترل اجرا ی برنامه می توانید از ساختار های **if—else** و **goto** استفاده کنید.

برای بلوک بندی نیز **T-sql**ساختار **BEGIN---END** را در اختیار برنامه نویسان قرار داده است.

مثال7:

```
IF  exists(set @Name=select Name from STD
                    Where s#=4585)
          Select 'The studentname is:',@Name
 Else
          Print 'no students found'
```

برای حلقه سازی نیز می توانید از ساختار **WHILE**با **CONTINUE**و**BREAK** استفاده کنید.

مثال8:

```
Declare @num1 smallint
Declare @num2 smallint
Set @num1=12
Set @num2=13
While(@num2!=20)begin
Select name,family,'has got the max grade
between',@num1,'and',@num2
from std
Where s# in(select max(s#) from reg where (grade between @num1
((and @num2
```

```
set @num1=@num1+1
set @num2=@num2+1
end
```

ازدیگر ساختار های پر کاربرد در نوشتن توابع جداول موقتی هستند که در آزمایش اول بحث شدند.

دسته بندی توابع **UDF**:

توابع**(UDF)** به سه دسته تقسیم می شوند :

- توابع اسکالر**(scalar)**
- توابع جدولی تک خطی**(inline)**
- توابع جدولی چند دستوری**(multistatement)**

توابع اسکالر

خروجی هایاین توابع تک مقداری می با شدکه معمولا برای انجام محاسبات به کار برده می شود

ساختارکلی تابع به صورت زیر تعریف می شود:

```
CREATE  FUNCTION [ owner_name. ] function_name
   ( [ { @parameter_name [AS] scalar_parameter_data_type [ =
default ] } [ ,...n ] ] )

RETURNS scalar_return_data_type

 [ AS ]

BEGIN
   function_body
   RETURN scalar_expression
END
```

مثال9 : تابعی می نویسیم که شماره یک دانشجو راگرفته و نام ونام خانوادگی وی را به فرمت خاصی (نام خانوادگی ،نام) نمایش می دهد .

```
Create FUNCTION fn_Nameret(@s# int)

 RETURNS NVARCHAR(37)

AS
```

```
BEGIN
      DECLARE @ret_Value  NVARCHAR(37)
      SELECT  @ret_Value=Name + '.' +Family
       FROM STD
      WHERE S# =@S#
       RETURN (@ret_Value)
End
```

تابع بالا را به صورت زیر احضار می شود.

```
select dbo.fn_Nameret(8008093)
```

مثال 10 :تابع ای می نویسیم که یک شماره دانشجویی وشماره یکی از ترم های تحصیلی وی رادریافت نموده و معدل آن ترم(**TrmGpa)** دانشجو را برگرداند**.**

```
create function fn_TrmGpa(@s# int,@TrmNo char(4))
returns dec(5,1)
begin
DECLARE @TotGrade  dec(5,1)
DECLARE @TrmRegUnit dec(5,1)
select
@TrmRegUnit=sum(CRS.Unit),@TotGrade=sum(REG.Grade*CRS.
Unit)
   from REG,CRS  WHERE
      REG.c# = CRS.c#
       and
```

```
        REG.TrmNo = @TrmNo
         and
        REG.s#=@s#
return(@TotGrade/@TrmRegUnit)
end
```

تابع فوق را می توان به شکل زیر استفاده کرد:

```
SELECT  dbo.fn_TrmGpa(8008093,'3802')
```

توابع جدولی تک خطی **(Inline table-valued function)**

خروجی این توابع یک جدول است . ساختار تابع به صورت زیر تعریف می شود:

```
CREATE FUNCTION [ owner_name. ] function_name
   ( [ { @parameter_name [AS] scalar_parameter_data_type [ =
default ] } [ ,...n ] ] )
RETURNS TABLE
[ AS ]
RETURN [ ( ] select-stmt [ ) ]
```

مثال 11 : تابع زیر با دریافت شماره درس و شماره ترم لیست نام و جنسیت دانشجویان آن درس را در ترم مذکور بر می گرداند .

```
Create function fn_myreg(@C#  char(7),@TrmNo char(4))
Returns table
```

```
As

Return (select STD.Name,STD.Family,CODEFILE.[Desc]  From
STD,CODEFILE

                Where     CODEFILE.Field = 'sex'

                    And

                      CODEFILE.Type = STD.Sex

                   And

                      exists(select * from REG

                              Where     (REG.C# =@C#

                                  and     STD.S#=REG.S#

                                   and      REG.Trmno=@Trmno)
)

        )
```

تابع فوق را می توان به شکل زیر استفاده کرد:

```
Select  *  from  fn_myreg (8006534,'3821')
```

می بینید که تفاوت اصلی این تعریف با توابع اسکالر در این است که نوع خروجی یک جدول تعریف شده است.محدودیتی که روی این نوع تعریف وجود دارد این است که خروجی تابع باید توسط یک دستور **select** ایجادشود.

توابع جدولی چنددستوری **(Multi-statement Table-valued Functions)**

در این نوع توابع این محدودیت که خروجی باید توسط یک دستور **select**ساخته شود وجود ندارد .

```
CREATE FUNCTION [ owner_name. ] function_name
   ( [ { @parameter_name [AS] scalar_parameter_data_type [ =
default ] } [ ,...n ] ] )
```

```
RETURNS @return_variable TABLE < table_type_definition >

 [ AS ]

BEGIN
   function_body
   RETURN
END

< table_type_definition > :: =
   ( { column_definition  |  table_constraint } [ ,...n ] )
```

مثال**12** : با استفاده از این روش تابعی می نویسیم که شماره یک دانشجو و شماره یک ترم وی را در یافت و در صورتی که معدل کل دانشجو بالاتر از 12 است شماره درس هایی را که دانشجو در آن ترم ثبت نام کرده است را بر می گرداند.در غیر این صورت حدول تهی برگرداند.

```
Create function Fn_crsnames(@S# int ,@TrmNo CHAR(4))

Returns  @crsnames table(c#  CHAR(7) not null)

As

Begin

Declare @mygpa  dec(5,2)

select @mygpa=Gpa from STD where S# = @S#

If @mygpa>12

   Begin

insert  @crsnames

Select  c#   from  REG

                Where  ( (S# = @S#) and (Trmno = @Trmno))

   End
```

return

End

از تابع فوق می توان به شکل زیر استفاده نمود:

Select * from Fn_crsnames(8008093,'3802')

تغییر وحذف تابع

برای تغییروحذف تعریف تابع(مانند دیگر اشیاءبه ثبت رسیده یک پایگاه داده) به ترتیب از دستور هـای **alter**و**drop** استفاده می کنیم .

ALTER function function _name . . .
DROP function function _name

نکته :چون در اینجا به انواع توابع اشاره کردیم بهتر است همین جا به تفاوت نیازهای اجرایی انواع توابع تفاوت آن ها با دیدها اشاره کنیم.گفتیم که برای محدود کردن دسترسی کاربران و برنامه های کاربردی به ستون ها یا سطر های مشخصی از جداول علاوه بر دید توابع و روالها می توانند گزینه های مناسبی باشند.در مورد دید می توانید بعد از ساختن دید با استفاده از **select** مجوز استفاده از آن را به کاربرن اهدا کنید.در مورد توابع وضعیت اندکی متفاوت است .برای استفاده از توابع اسکالر به مجوز **Execute** نیاز دارید .برای استفاده از توابع جدولی که خروجی آن از نوع جدول است به مجوز **select** هم نیاز دارید .

دستور کار

وارد محیط **Query Analyzer** شوید هریک از پرسشهای زیر را با استفاده از دستورات **T-SQL** پاسخ داده و اجرا کنید**.**

(همه اسکریپت های نوشته شده را ذخیره نمایید.)

1 - دانشجویانی را که بدون رعایت پیش نیازی در دروسی ثبت نام نموده اند ، استخراج کرده ،شماره و نام هر دانشجو همراه با شماره و نام درس ثبت نام شده را نشان دهید .

2- دانشجویانی را که بدون رعایت هم نیازی در دروسی ثبت نام نموده اند ، استخراج کرده ،شماره و نام هر دانشجو همراه با شماره و نام درس ثبت نام شده را نشان دهید .

3- با احضار تابع مرحله قبل **TrmGpa** را برای ترم آخر دانشجو در فایل **STDTRM** اصلاح کند .

4- یک تابع به نام **fn1_StdGpa** تعریف کنید که شماره یک دانشجو را دریافت نموده و با احضار تابع مرحله قبل ، معدل کل **(Gpa)** دانشجو راودر صورتی که چنین دانشجویی وجود نداشت مقدار**1-** را برگرداند .

5- یک تابع به نام **fn2_StdGpa** تعریف کنید که فقط با استفاده ازجداول **CRS**و**REG** شماره یک دانشجو را دریافت نموده و معدل کل **(Gpa)** را محاسبه کرده و ، در صورتی که چنین دانشجویی وجود نداشت مقدار**1-** را برگرداند .

6- با دریافت شماره دانشجویی و احضار تابع مرحله 5 **Gpa** را در فایل**STD** اصلاح نماید .

7- با استفاده از محیط **Enterprise Manager** یک تابع به نام**fn_TotPasUnit** ایجاد کنید که شماره دانشجویی را دریافت نموده و مجموع واحد های پاس شده را برگرداند ، در صورتی که چنین دانشجویی وجود نداشت مقدار**1-** را برگرداند.

8 - با استفاده ازتب **Template**تابعی با نام**fn_TotRegUnit** ایجاد کنید که شماره دانشجویی را دریافت نموده مجموع واحد های ثبت شده **(TotRegUnit)** را برگرداند. در صورتی که چنین دانشجویی وجود نداشت مقدار**1-** را برگرداند.

9- با احضار توابع مراحل 7و8 فیلد های **TotPasUnit** و**TotPasUnit** را در جدول**STD** ا اصلاح نمایید .

10- یک تابع به نام **fn_RegCtrl** بنویسید که یـک شماره دانشجویی را دریافت نموده وصحت ثبت نام وی را بررسی نموده و به عنوان نتیـجه یـکی از مقادیـر 0 تا 3 را به شرح زیر برگرداند .

- **0**، اشکال وجود ندارد .
- 1 ، یـعنی مشکل پیش نیاز وجود دارد .
- 2 ،یـعنی مشکل هم نیازی وجود دارد .
- 3 ، یعنی مشکل پیـش نیـــاز ی و هم نیازی وجود دارد .

11- ا استفاده رویه ذخیره شده **sp_helptext** تابع نوشته شده در مرحله قبل را مشاهده نمایید .

12- نام دروس سه واحدی گروه کامپیوتر که دانشکده برق و کامپیوتر ارائه می کند را استخراج کنید .

آزمایش 3

رویه هادر**SQL-Server**

پیش آگاهی

مقدمه

رویه ها نیز مانندتوابع ابزارهای مناسبی برای دسته بندی دستورات پرکاربرد هستند. رویه ها حتی می توانند مقادیر خروجی داشته باشند ولی تفاوت عمده این دوساختاردرنحوه احضارآن ها برای گرفتن مقداربازگشتی است . احضار توابع صریح است بدین معنی که در احضار توابع می توان مقداری را مساوی با تابع قرار دادو بعداز احضار متغیر با مقدار بازگشتی تابع مقدار دهی می شود. ولی برای گرفتن خروجی از رویه ها باید یک یا چندین پارامتر را به عنوان مقادیر خروجی معرفی کنیم و رویه بامقدار گذاری و تغی‌ی‌ر ای‌ن پارامترها مقدار خرجی را**Set**می کند.

تفاوت های اندک دیگری نیز بین این دو ساختار وجود دارد که در حین بررسی نحوه تعریف رویه ها به آن ها می پردازیم.

بعد از تعریف رویه ها ،آن ها معمولا توسط بخش بهی‌نه ساز **SQL-SERVER**ی‌ک بار کامپای‌ل شده و بهتری‌ن مسی‌ر اجرای‌ی برای آن ها ساخته می شود.

هنگامی که ی‌ک رویی‌ه احضار می گردد تنها کاری که **SQL SERVER**انجام می دهد جای‌گزی‌نی پارامترهای فرستاده شده در رویی‌ه واستفاده از طرح اجرای‌ی از پی‌ش تهی‌ه شده**(Catch)**(بدون کامپای‌ل وبهی‌نه سازی مجدد آن)برای اجرای احضار رویی‌ه است.

ساختارتعریف رویه ها :

```
CREATE  PROC [ EDURE ]  procedure_name
   [ { @parameter data_type }
      [ = default ] [ OUTPUT ]
   ] [ ,...n ]

{ WITH  RECOMPILE}
AS sql_statement [ ...n ]
```

بررسی پارامتر های تعریف فوق:

- دیده می شودکه در تعریف رویه ها نیز مانند توابع بعد از نام رویه لیست پارامتر های آن رویه می آیدکه این پارامتر ها می توانند ورودی ،خروجی و یا ورودی-خروجی باشند .
- با استفاده ازکلمه کلیدی **Default** می توان مقداری را برای یک پارامتر تعیین کرد که در صورتی که در حین احضار رویه این آرگومان احضار نگردد با مقدار پیش فرض جایگزین شود .(توجه کنید که در این صورت لازم است تااحضار رویه با تکنیک **Call by name**صورت گیرد .)
- کلمه کلیدی **Output** بدین معنی است که پارامتر مورد نظر می تواند به عنوان خروجی یا ورودی– خروجی مطرح با شد.
- گفتیم که رویه های ذخیره شده فقط یکبار کامپایل می شوند .اگر بخواهیم که رویه در هر بار اجرا کامپایل شود ودر نتیجه نسبت به تغییرات ایجاد شده در پایگاه داده حساس باشد ،(مثلا اضافه شدن یک شاخص) می توانید از عبارت**WITH RECOMPILE** استفاده کنید.

مثال1: رویه ای بنویسید که شماره یک دانشجو وشماره یک ترم وشماره یک درس را دریافت ودرس مورد نظر را در ترم مذکوربرای آن دانشجو ثبت نام کند. این رویه پارامتر چهارمی را هم دریافت می کندکه به عنوان خروجی مجموع واحد های ثبت نام شده وی در ترم مذکوررادرآن برمی گرداندو در صورت داشتن هر گونه مشکل برای ثبت نام این درس پارامتر چهارم به شرح زیر مقداردهی می شود:

- **(-1)** دانشجو در آن ترم ثبت نام ندارد.
- **(-2)** دانشجو مشکل پیش نیازی برای ثبت نام درس دارد.
- **(-3)** دانشجو مشکل هم نیازی برای ثبت نام درس دارد.

```
Create proc pr_reg
@s# int,
@trmno char(4) default 3831,
@c# int ,
@trmregunit dec(3,1)output
As
if  exists(select * from stdtrm where s#=@s# and trmno=@trmno)
      if not exists (select cp# from prereq

                                                  where ( c#=@c# and not
exists(select c# from reg
                                                     where reg.s#=@s#
                                                     and
reg.c#=prereq.cp#))


if not exists(select cc# from coreq
                  where  c#=@c# and not exists)select c# from reg
                                                     where reg.s#=@s#
                                                      and
reg.c#=coreq.cc#))
                begin
                  insert reg(s#,c#,trmno)  values(@s#,@c#,@trmno)
                  set @trmregunit=sum(unit) from crs inner join reg
                                    on crs.c#=reg.c#
                                      Where          s#=@s#          and
trmno=@trmno)
                end
        else
             @trmregunit=-3.0
      else
          @trmregunit=-2.0
else

 @trmregunit=-1.0
```

روش های احضار رویه ها در**SQL SERVER**:

دو روش برای احضار رویه ها وجود دارد در روش اول بعد از نام رویه لیست پارامتر هابا یک کاما بین آن ها می آید و اگر پارامتری را نخواهیم بفرستیم جای آن را خالی می گذاریم .

مثال**2** :با این روش رویه**pr_reg** را احضار می کنیم:

Declare @myvar dec(3,1)
pr_reg
8014681,1122323,@myvar

مثال 3 :پیاده سازی روش دوم که آن رافراخوانی بانام نیز می نامیم را در مثال زیر می بینید:
(Call By Name)

Declare @myvar dec(3,1)

pr_reg
@s#=8014681,
@c#= 11223323,
@trmregunit=@myvar

تغییر وحذف رویه های ذخیره شده

برای تغییروحذف یک رویه ذخیره شده (مانند دیگر اشیاءبه ثبت رسیده یک پایگاه داده) به ترتیب از دستور های

Alterو**Drop**استفاده می کنیم.

مثال4 : در مثال زیر رویه**Pre-reg**راتغییر می دهیم:

```
Alter proc pr_reg(
@name varchar(16),
@family varchar(20),
@trmno char(4),
@c# int,
@trmregunit dec(3,1) output

As ……
```

بررسی چند نکته :

1. می توان از عبارت **with recompile** درجایی که یک رویه احضار می شوداستفاده کرد. در این صورت رویه دوباره کامپایل وبهینه سازی می شود .

مثال5 :

```
Execute pr_reg
@S#= 8014681,@TRMNO=3822,@C#=1116554,@trmregunit
output

with recompile
```

2. احضارویه به صورت صریح وبه کمک کلمه کلیدی **EXECUTE** انجام می شود .(درصورتی که احضار درابتدای دسته انجام گیرد ، نیازی به نوشتن این کلمه نیست .)

```
Declare @trmregunit dec(3,1)
Execute pr_reg  8014681,3822 ,1116554,@trmregunit output
```

مثال6 : می خواهیم رویه ای بنویسیم که دروس یک دانشجورا که بدون رعایت هم نیازی درترم جدیدثبت نام کرده ، حذف نماید .

```
create procedure pr_delete_coreq_reg (@st# int)
as
begin
delete from reg  where
```

```
reg.s# = @st#
and
reg.c#  in (select  coreq .c# from coreq
where
coreq.c#  = reg.c#
and
reg.s#  = @st#
and
coreq.cp#  not in (select  reg. c#  from  reg
where
reg.s#  =  @st#     ))
end
```

رویه فوق را می توان به شکل زیر احضار نمود.

```
execute  pr_delete_coreq_reg  8004367
```

دستورکار

1- یک رویه به نام**Pr_ TrmGpa** بنویسید که شماره یک دانشجو و شماره ترم را دریافت نموده و معدل ترم دانشجو را محاسبه و بر گرداند .
2- یک رویه به نام**Pr_tot_stdtrm** بنویسید که شماره یک دانشجو را دریافت نموده ومعدل کل،کل واحدهای گذرانده وکل واحدهای اخذشده دانشجورا محاسبه نموده وبرگرداند .
3- یک رویه به نام **Pr_Delete_Prereq_reg** بنویسید که شماره دانشجویی را گرفته ودروسی راکه دانشجو بدون رعایت پیش نیازی ثبت نام نموده ، حذف نماید .
4- رویه ای به نام **Pr_Mydelete** بنویسید که شماره درس و حداقل تعداد ثبت نام در یک درس برای تشکیل شدن آن درس را دریافت و در صورتی که نعداد ثبت نام در درس به حد نصاب نرسید ه است کلیه ثبت نام های فعلی درس را حذف کند .

آزمایش4

(CURSORS) کرسرها

پیش آگاهی

مقدمه

باتوجه به اینکه خروجی دستور**SELECT**معمولا یک مجموعه (جدول)بوده ودرزبان های برنامه نویسی ، امکان کارکردن باجداول وجودندارد، به کمک کرسرها امکان دسترسی به رکوردهای یک جدول به شکل رکوردبه رکورد وجوددارد .

قبل از معرفی نحوه تعریف یک کرسر و خصوصیات آن با توجه به زیاد بودن دسته بندی های مختلف کرسرها برحسب خصویات مختلف آن بهتر است ابتدا به معرفی بعضی ویژگی هایی کرسر ها بپردازیم :

1- طول عمر(حوزه)کرسر:

کرسر هایی که ساخته می شوند به صورت پیش فرض تا آخر پایدار بودن اتصال جاری به سرور باقی می مانند مگر این غیرفعال **DEALLOCATE**که صراحتا از بین برده شوندیا توسط عبارت شوند.

2- قدرت تغییر و به روز آوری جداولی که کرسر روی آن ها حرکت می کند:

به این معنی که بعد از این که کرسری ساخته شد آیا می تواند تغییری در تاپلی که روی آن قرار دارد ایجاد کندیا این که فقط حق خواندن ازاین تاپل را دارد.

3- نحوه حرکت کرسر روی تاپل های جدول :

این که آیا کرسر این قدرت را دارد که مثلا بتواند علاوه بر جلو رفتن به سمت عقب هم حرکت کندیا این که به ابتدای جدول پرش کند.

4- حساسیت به تغییرات ایجاد شده در جدول :

به این معنی که آیاپس ازبازکردن کرسر، اگرتغییراتی درجداول منبع کرسربه وجودبیاید،درکرسرتاثیر دارد یاندارد .

5- سرعت ایجاد و حرکت کرسر:

این خصوصیت یک خصوصیت تر کیبی است و با توجه به این که یک کرسر کدام یک از خصوصیات موارد قبل را داشته باشد تعیین می شود .مثلا مسلما کرسری که هم رو به جلووهم روبه عقب حرکت می کند از کرسری که تنها رو به جلو می رود کند تر است.

نحوه استفاده از کرسرها

برای استفاده از یک کرسر باید قدم های زیر را طی نمود :

1- تعریف کردن کرسر: تعریف کرسربه شکل کلی زیرانجام می شود :

```
DECLARE cursor_name CURSOR
[ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
FOR select_statement
[ FOR UPDATE [ OF column_name [ ,...n ] ] ]
```

بررسی بعضی پارامتر های تعریف فوق :

- کلمه کلیدی **Local** ، میدان استفاده از کرسر را به تابع ،رویه یا رهانایی که رویه در آن تعریف شده محدود می کند .کلمه کلیدی **Global** به این معناست که کرسر تا پایدار بودن اتصال جاری به سرور باقی می ماند مگر این که صراحتا از بین برده شود .
- **FORWARD_ONLY :** به این معنی است که کرسر می تواند تنها روبه جلو حرکت کند در حالی که **SCROLL** بودن کرسرهم حرکت رو به جلو وهم حرکت روبه عقب را برای کرسر ممکن می کند .
- **STATIC** : این کلمه کلیدی باعث می شود که کرسر اصلا نسبت به تغییرات ایجاد شده در پایگاه داده حساس نباشد. زیرا در این حالت کرسر یک کپی از داده هارا به **tempdb**منتقل می نماید .
- **KEYSET** : در این حالت فقط ستون های کلیدی تاپل های کرسر را مشخص می کنند(این مجموعه را **KEYSET** می نامند) در **tempdb** نگهداری می شوند.بعد از باز کردن کرسر مقادیر این کلید ها چون کپی از جدول اصلی است نسبت به تغییرات جداول اصلی غیر حساس می شوندولی بقیه ستون ها نسبت به تغییرات حساسند .
- **DYNAMIC** : در این نوع کرسرهابا هر بار حرکت درکرسردستور**SELECT** آنها دوباره اجرا می شود بنابراین کاملا نسبت به تغییرات پایگاه داده ها حساس است .

- **FAST_FORWARD** : این کرسر ها سریع ترین کرسرها هستندوترکیبی از گزینه های **FORWARD _ ONLY** و**READ_ONLY** هستند .
- **SCROLL_LOCKS** : در این حالت بر خلاف حالت **READ_ONLY** امکان **UPDATE** وجود دارد. اینجاکلیه تاپلهای کرسر(ونه فقط تاپلی که جاری است) قفل می شودوهیچ شی **(Object)** جز خود همین کرسرنمی تواند آن ها را تغییر دهد.در این حالت مطمئنیم که **INSERT**و**UPDATE**با مشخص کردن تاپل مورد نظر حتما با موفقیت انجام می شود .
- **OPTIMISTIC** : تفاوت این مورد با **SCROLL_LOCKS**در قفل نکردن داده هااست .
- یادآور می شویم امکان استفاده از بعضی گزینه های فوق که معمولا منطقاً متضاد هستند به صورت هم زمان وجود نداردبه عنوان مثال یک کرسر از نوع **FAST_FORWARD** نمی تواند خاصیت**SCROLL _LOCKS** یا **OPTIMISTIC** راداشته باشد .

2- باز کردن کرسر:

OPEN { [GLOBAL] cursor_name }

در صورتی که دو کرسر همنام هم زمان هم به صورت محلی و هم به صورت سراسری وجود داشته باشند برای باز کردن کرسر سراسری بایدکلمه کلیدی **Global**استفاده کرد .

3- خواندن سطور :

اجرا می گردد در واقع یک رکورد جدید از کرسر خوانده می شود .**FETCH**هر بار که دستور

```
FETCH
    [ [ NEXT | PRIOR | FIRST | LAST
        | ABSOLUTE n
        | RELATIVE n
      ]
      FROM
    ]
{ { [ GLOBAL ] cursor_name } }
[ INTO @variable_name [ ,...n ] ]
```

برای کنترل واکشی سطرها درهنگام استفاده از کرسرمتغیرسیستمی **@@FETCH_STATUS** به کاربرده می شود . مقدار خروجی این تابع وضعیت آخرین دستور **FETCH** را نمایش می دهد،در صورتی که واکشی موفقیت آمیز باشداین تابع مقدار صفر را بر می گرداند.

- کلمه کلیدی **NEXT** باعث می شود تا سطربعداز سطر جاری واکشی شود.کلمات کلیدی **FIRST**و**PRIOR**و**LAST** به ترتیب معادل اولی ، قبلی و آخرین می باشند.
- **ABSOLUTE n** سبب می شود تاپل **n**ام از ابتدای کرسر واکشی شود .
- **RELATIVE n** سبب می شود تاپل **n**ام بعد از تاپل جاری کرسرواکشی شود.
- با استفاده از عبارت **INTO @variable_name [,...n]** متغیر های ذکر شده بعد از **INTO**بانتایج حاصل از واکشی کرسر مقداردهی می شوند.انواع داده ای این متغیر ها بایدبا انواع داده ای ستون های ذکر شده در جلوی **SELECT**کرسر به ترتیب یکسان باشند .

4- بستن کرسر:

CLOSE { [GLOBAL] cursor_name }

5- رها کردن حافظه اشغال شده توسط آن

DEALLOCATE { [GLOBAL] cursor_name }

قدم های 2 تا4 می تواند تکرار شود یعنی یک کرسر را بعد از بستن دوباره باز کرد که در این صورت کرسر مجدداً از جدول اصلی مقدار دهی می شود.

مثال **1**:حذف دروس ثبت نام شده دانشجویان که بدون رعایت هم نیازی اخذ نموده اند با استفاده از آزمایش 3 . (با فرض این که هر درس فقط یک درس هم نیاز **6**یک کرسر و رویه نوشته شده در مثال دارد.)

ابتدا کرسر مورد نظر را ایجاد می نمائیم:

```
declare cr_delete_prereq_reg cursor
SCROLL_LOCKS
FOR select s# from std
open cr_delete_prereq_reg
declare @st#   int
fetch next from  delete_prereq_reg into  @st#
while ( @@fetch_status = 0)
```

```
begin
execute pr14   @st#
fetch next from  cr_delete_prereq_reg into  @st#
end
close cr_delete_prereq_reg
deallocate      cr_delete_prereq_reg
```

یکی از روش های ارتباط با رویه ها این است که یک کرسر به عنوان پارامتر خروجی معرفی شود.در واقع خروجی رویه یک کرسر است که می توان با آن کار کرد .

مثال2 : می خواهیم رویه ای بنویسیم که شماره دانشجو را به عنوان ورودی دریافت نموده و نام دروسی که دانشجو ثبت نام نموده در پارامتر خروجی که یک کرسر است قرار بدهیم .

```
CREATE PROCEDURE pr_stdcourses_cursor
@s# int,
@stdcourses_cursor CURSOR VARYING OUTPUT
AS
SET @stdcourses_cursor = CURSOR
OPTIMISTIC
DYNAMIC FOR
SELECT cname from
FROM reg inner join crs
 On crs.c#=reg.c#
Where reg.s#=@s#
OPEN @titles_cursor
```

کلمه کلیدی **VARYING** را در مواردی که یک کرسر به عنوان پارامتر خروجی تعریف می شود به کار برده می شود . در رویه بالاابتداخواص کرسر**@stdcourses_cursor** راکه پارامتر خروجی این رویه است را **set**کرده ایم .

سپس این کرسر را باز و آماده واکشی کرده ایم . در دستورات زیر از این رویه استفاده می کنیم.

```
DECLARE @MyCursor CURSOR
EXEC pr_stdcourses_cursor @stdcourses_cursor = @MyCursor OUTPUT
Declare @cname varchar(25)
WHILE (@@FETCH_STATUS = 0)
```

```
BEGIN
  FETCH NEXT FROM @MyCursor into @cname
Print 'the next course is:'+@cname
END
CLOSE @MyCursor
DEALLOCATE @MyCursor
GO
```

دستورکار:

5- بااستفاده ازرویه **Pr_ TrmGpa** دردستورکارآزمایش 3ویک کرسر به نام **Update_CR_TrmGpa** ، معدل ترم **(TrmGpa)** دانشجویان را اصلاح نمایید .

6- بااستفاده ازرویه**Pr_tot_stdtrm** دردستورکارآزمایش 3ویک کرسر به نام **Cr_tot_stdtrm**معدل کل **(Gpa)** ، کل واحدهای گذرانده **(TotpassUnit)** وکل واحدهای اخذشده**(TotRegUnit)** دانشجویان را اصلاح کنید .

7- با استفاده از رویه **Pr_Delete_Prereq_reg** دردستورکارآزمایش 3و یک کرسربه نام **Cr_Delete_Prereq_reg** دروس دانشجویانی را که بدون رعایت پیش نیازی (در جدول **REG**)ثبت نام کرده اند حذف نماید .

8- با استفاده از رویه**Pr_Mydelete** دردستورکارآزمایش 3و یک کرسربه نام **Cr_Mydelete** ثبت نام های کلیه دروس به حدنصاب نرسیده را حذف کند .

9- آیا می توان کرسرهای **Dynamic** را با استفاده از کرسرهای **Static** شبیه سازی کرد؟بررسی کنید .

آزمایش **5**

تراکنش هاوتریگرها در**SQL**

پیش آگاهی

مقدمه :

یکی ازاصول تامین جامعیت داده ای ، امکان اعمال به روز رسانی منتشرشونده وتعریف واحدهای منطقی کاردرراستای اجرای کامل یاعدم اجرای کلیه مواردازیک مجموعه کارمی باشد .
برای پیاده سازی این اصل **SQL-Server** تریگر(رهانا)وتراکنش رادراختیاربرنامه نویس قرارداده است . احضارتریگرالزاما به صورت ضمنی بوده وازطریق اجرای یکی ازدستورات **delete، insert** یا**update** انجام می شود . برنامه نویس رهانارابرای اجرای دستوراتی که بایدبه صورت خودکاربرای حفظ جامعیت داده ها انجام شود ، تعریف می کند . بنابراین تریگررامی توان تراکنشی دانست که باتغییرات داده فعال می شود وامکان اجرای یک واحد منطقی کاررا درجریان تغییرات داده ای فراهم می کند .

تراکنش **(Transaction)**

تراکنش مجموعه ای از دستورات **SQL** است که معمولا تغییری در پایگاه داده ایجاد می کند به گونه ای که جامعیت و یکپارچگی داده ها همچنان برقرار باشد.این مجموعه دستورات یا به طورکامل اجراء می گردند ویا درصورت وجود اشکال دربین دستورات اجرای تمامی آن ها لغومی گردد .

درواقع تراکنش ،یک واحد منطقی کار**(Logical unit work)** است که با استفاده از آن می توان از پایان درست وکامل کار اطمینان پیدا کرد.
SQL برای اجرای تراکنش ها از فایل ثبت تراکنش ها **(Log file)** استفاده می کند که کلیه تغییرات درآن ذخیره می شود. برای افزایش سرعت وهمچنین برای اطمینان ازاجرای درست وکامل تراکنش ،معمولا تغییرات موردنظرعلاوه برآن که درفایل های اصلی اعمال می شود ، درفایل **log** نیزنوشته شده والبته رکوردهای تغییرداده شده درحالت **lock** قرارمی گیرند . درانتها درصورتی که قرارباشد تغییرات برگشت داده شود، ازفایل **log** استفاده شده وتغییرات برگشت داده شدهودرنهایت به هرحال رکوردهای **lock** شده آزادخواهند شد .
دستورات کنترل تراکنش
Begin Transaction : نقطه شروع یک تراکنش است که ساختارکلی آن به شکل زیراست :

BEGIN TRAN [SACTION] [transaction_name | @tran_name_variable
[WITH MARK ['description']]]

transaction_name : نامی اختیاری است که به تراکنش داده شود وحداکثر32 کارکترمی باشد. درتراکنش هایی که به صورت تودرتونوشته می شود، بهتراست به بیرونی ترین تراکنش نامی اختصاص داده شود . **@tran_name_variable** : نام یک متغیرتعریف شده توسط کاربراست که بایدبایکی ازانواع داده ای **varchar، nchar ، char** ویا**nvarchar** قبلا تعریف شود .

دفترتراکنش

دردفترتراکنش**(Transaction Log)** ،تراکنش بانامی که بعداز**begin tran** آورده شده ،همچنین نام پایگاه داده ای که تراکنش برروی آن تعریف شده ، نام کاربر ،تاریخ ، زمان ،شرح تراکنش وشماره ترتیب **Log (LSN)** ثبت می شود . برای هرتراکنش که **mark** می شود یک رکورددرجدول **logmarkhistory** در**msdb** درنظرگرفته می شود . اگریک تراکنش **mark**شده، تغییراتی درپایگاه داده ایجادکرد وکاربربخواهدکه تغییرات انجام نشود ،برای بازسازی پایگاه به وضعیت آن در یک زمان وتاریخ مشخص کافی است بادادن یکی ازمشخصات تراکنش ،آن داده هارادوباره **restore** کرد .

WITH MARK ['description']: اگراین عبارت استفاده شود، بایدبرای تراکنش نامی بیان شده باشد . کاربرمی تواند بدین وسیله بادادن نام تراکنش ، آن رادردفترتراکنش**(Log**

Transaction)ثبت کند.
description : دراین قسمت توضیحی درموردتراکنش می تواندآورده شود .

Commit Transaction : به ازای هر**begin transaction** حداقل یک **commit transaction** وجوددارد وپایان منطقی یک تراکنش رابیان می کند . پس ازاجرای این دستورهمه تغییراتی که ازشروع تراکنش تااین قسمت درپایگاه داده بایدانجام شود،دائمی شده و منابعی که تراکنش دراختیارگرفته آزادمی گردد .

COMMIT[TRAN [SACTION] [transaction_name | @tran_name_variable]]

transaction_name و**@tran_name_variable** : نام هایی هستند که بعداز **begin tran** آورده می شود .
اگر ترکنش هابه صورت تودرتوتعریف شده باشند ، **commit** ای که درونی تعریف شده تاوقتی که بیرونی ترین **commit tran** اجراءنشود ، منابع انحصاری راآزادنمی کند .
Save Transaction : این عبارت یک نقطه را دربین تراکنش برای استفاده دردستور **Rollback transaction**تعیین می کند .

SAVE TRAN [SACTION] { savepoint_name | @savepoint_variable }

پارامترها همانند قبل تعریف می شوند .

ROLLBACK TRANSACTION : این دستور، وضعیت داده های پایگاه رابه نقطه شروع تراکنش ویا به نقطه ای که توسط **savepoint** ،که با **SAVE TRANSACTION** مشخص شده ، به عقب برمی گرداند یعنی تغییرات ایجادشده ازابتدای تراکنش یاازsavepoint تااینجاراخنثی می کند . اگراین دستوربدون نام آورده شود ، تاشروع تراکنش تغییرات رابه عقب برمی گرداند واگردریک تراکنش تودرتوبه کاررود،تمام تراکنش های داخلی راهم عقبگردمی کندتابه بیرونی ترین **BEGIN TRAN** برسد . اگر دردرون تراکنش چند **savapoint** وجودداشت **ROLLBACK** تانزدیک ترین نقطه **savepoint** به **ROLLBACK** ، عقبگرد می کند .
transaction_name و**@tran_name_variable** : دقیقا مانند **begin tran** تعریف می شوند .
savepoint_name : نامی است که در **SAVE TRANSACTION** تعریف شده است وباعث می شود که **ROLLBACK TRAN** تا این نقطه به عقب برگردد .

@savepoint_variable : نام متغیری است که در **SAVE TRANSACTION** مشخص می شود وقبلا بایدتعریف شده باشد .

@@Trancount : متغیری سراسری است که مقدار آن ازنوع **integer** است ومعرف تعدادتراکنش های فعال می باشد . باهر**BEGIN TRANSACTION** یک واحد به آن اضافه می شود وباهر**COMMIT TRAN** یک واحدازآن کم می شود .**ROLLBACK TRAN** مقدار آن راصفرمی کند به جز**ROLLBACK TRAN savepoint_name** که تغییری درمقدار آن ایجاد نمی کند .

@@error: یک متغیرسراسری است ومقدارآن معرف این است که آخرین دستوراجراشده باموفقیت انجام شده یانه .این متغیر یک مقدار**integer** رابرمی گرداندکه کد خطایی که دراجرای آخرین دستور **T_SQL** انجام شده، رخ داده است را برمی گرداند . که در صورت عدم وجود خطا ،مقدار صفرمحتوی آن می باشد .

متن خطا همراه باشماره ای که مشخص کننده نوع خطاست درجدول سیستمی **sysmessages** آورده شده است .

مثال 1 : می خواهیم یک مجموعه دستور بنویسیم که در ابتدا دانشجویی را در ترم خاص ودردرس خاص ثبت نام نماید ، در صورتی که مجموع واحد های ثبت نام شده برای دانشجوکمتر از 100 واحد باشد ، ثبت نام وی را در این درس لغو و در صورتی که معدل کل وی کمتر از 10 باشد ثبت نام وی را در این ترم لغو نماید.

```
begin Transaction
Insert into STDTRM values (8006530,'3811',null)
Save Transaction beforeregiser
insert into REG values(8006530,'1117340','3811',null)
if (select
      TotRegUnit  from STD
            where STD.S# = 8006530)<100)
begin
  print ' You can not Register in this course '
  Rollback Transaction beforeregiser
end
else if(select Gpa from STD WHERE STD.s#=8006530)<10
begin
  print ' You can not Register in this term '
  Rollback Transaction
```

```
end
PRINT 'success'
Commit Transaction
```

تریگر(TRIGGER)

تریگر(رهانا) نوع خاصی از رویه است که توسط کاربر درراستای تامین جامعیت درپایگاه داده هاتهیه شده ودرزمان تغییر پایگاه ازطریق درج ، حذف واصلاح یک **table**یا **view** فعال شده وعملیات موردنظررا انجام می دهد . بدین ترتیب می توان باتعریف یک تریگراز اعمال تغییرات غیرمعتبریاناسازگار در پایگاه داده ها جلوگیری کرده وازجامعیت ودرستی داده ها اطمیینان حصل نمود . این رویه پارامتر ندارد و به طور ضمنی فعال می شود یعنی همانند توابع و رویه ها به صورت مستقیم احضار نمی شوند بلکه در ضمن اجرای یکی ازدستورات **Update,Insert** یا **Delete** اجرا می شوند ، بنابراین هیچ دستوری وجود ندارد که بتوان با استفاده از نام تریگر آن تریگر را احضارنمود و تنها عامل فعال کردن آن تغییرات داده می باشد.

برروی هر یک ازدستورات **insert**،**delete** و**update** می توان چند تریگر تعریف کرد .اگردراجرای بخشی ازتریگراشکالی ایجادشود، از کلیه عملیات انجام شده صرفنظر می شود، به این معنا که همواره یا کل عملیات انجام شده ویا هیچ عملی انجام نخواهدشد .

همچنین می توان تریگرهاراباعبارات**ENABLE TRIGGER** و **DISABLE TRIGGER** که در**ALTER TABLE** به کاربرده می شوند ، به ترتیب فعال یاغیرفعال کرد .

جداول مجازی **Deleted** و **Inserted**

این جداول ساختار مشابه با جدولی دارد که در آن عمل درج ، حذف ، ویا اصلاح انجام شده است و تریگر روی آن تعریف شده است وبه صورت خودکار توسط **SQL** ایجاد می شوند. وجود این دو جدول نیاز به تعریف متغیر برای نگهداری اطلاعات را برطرف می کند .اگر رکورد جدیدی اضافه شود این رکورد هم در جدول اصلی وهم در جدول **Inserted** وارد می شود و اگر رکوردی حذف گردد آن رکورد وارد جدول **Deleted** نیز می شود. در هنگام **Update** نیز رکورد قبلی در جدول **Deleted** و رکورد جدید در جدول **Inserted** قرار می گیرد. در واقع برای هر جدولی که تریگر تعریف شده است این دو جدول در هنگام فعال شدن تریگر به صورت پویا برای هر کاربر در حافظه **RAM** سیستم ایجاد می گردد .

به تاپل های این دو جدول مانندتمام جداول دیگر به کمک دستور های **SQL** می توان دسترسی داشت ولی مستقیما نمی توان آن را تغییر داد.

ایجاد تریگر ها

ساختاراصلی یک تریگر به صورت زیراست :

```
CREATE TRIGGER trigger_name
ON { table | view }

{
  { { FOR | AFTER | INSTEAD OF } { [ INSERT ] [ , ] [ UPDATE ] }
    AS
    [ { IF UPDATE ( column )
      [ { AND | OR } UPDATE ( column ) ]
        [ ...n ]
       } ]
    sql_statement [ ...n ]
  }
}
```

توجه شود که جدول یادیدی که تریگربرروی آن تعریف می شود ، بعدازکلمه **ON** بیان می شود.

AFTER: یعنی این تریگر بعد از اعمال تغییرات دادهای در جدول مورد نظر فعال گردد.

INSTEAD OF: یعنی تغییرات داده ای در جدول یا دید مورد نظر اعمال نشده و فقط تریگر فعال گردد. البته ممکن است تغییرات بعداً از طریق دستورهای تریگر در جدول یا دید مورد نظر اعمال شود.

FOR :مانند**AFTER** عمل می کند بااین تفاوت که می تواند روی دید ها هم تعریف شود .

تذکر: تریگر درصورت مشخص نکردن هیچ کدام ازسه مورد بالا ، **AFTER** راپیش فرض قرار می دهد .

پس ازکلمه **AS** بدنه تریگر قرارمی گیرد که درآن عملی که تریگر قرار است انجام دهد ، نوشته می شود .

update(column) : مقدار این عبارت**true** یا **false** بوده و نشان می دهد که آیا روی **column** مورد نظر عمل اصلاح انجام شده است یا نه (در هنگام درج و اصلاح فیلد مورد نظر نیز این مقدار**true** می باشد. به کمک این عبارت و دستور**if** می توانیم در مقابل تغییر بعضی از فیلدهای مورد نظر عکس العمل مناسب پیش بینی نمود.

نکته :دربدنه تریگرنمی توان ازدستورات **CREATE DATABASE** ، **ALTER DATABASE** و **DROP DATABASE** استفاده کرد .

تغییر و حذف تریگر

برای تغییرویاحذف یک تریگربه ترتیب ازدستورات **ALTER TRIGGER** و **DROP TRIGGER** استفاده می شود .

```
ALTER TRIGGER  trigger_name      . . .
DROP TRIGGER trigger_name
```

مثال **2** : باتوجه به این که به دلیل قانون جامعیت ارجاعی نمی توان دانشجویی که ثبت نام دارد را از جدول **STD** حذف نمود تریگری روی این جدول می نویسیم که به جای دستور **Delete** ابتدا دروس ثبت نامی وی در جدول **REG** وبعدا رکورد های مرتبط با دانشجو در جدول **STDTRM** را حذف نموده ونهایتا دا نشجو را از جدول **STD** حذف نماید.

```
create trigger Tr_Delete_STD
ON STD
INSTEAD OF DELETE
AS
Begin
DECLARE @s# int
select @s# = s#  from Deleted
delete from REG where REG.S# = @S#
delete from STDTRM where STDTRM.S# = @S#
delete from STD where STD.S# = @S#
print ' Delete student from REG and STDTRM and STD '
End
```

مثال3 : تریگری می نویسیم که وقتی نمره دانشجو در جدول **REG** اصلاح می شود معدل ترم دانشجو**(TrmGpa)** در جدول **STDTRM** اصلاح شود .

```
CREATE Trigger Tr_Update_TrmGpa
ON REG
after Update
AS
if Update(Grade)
BEGIN
declare @SumGrade dec(4,1)
declare @SumUnit dec(4,1)
DECLARE @S# INT
DECLARE @TrmNo char(4)
DECLARE @c# char(7)
SELECT @S# = S# , @TrmNo = TrmNo , @C# = C# FROM
INSERTED
```

```
select @SumGrade = sum(CRS.Unit*REG.Grade),@SumUnit = sum
(unit) from REG,CRS
   where REG.c# = CRS.c#
      and
       REG.C# = @C#
      and
       REG.TrmNO =@TrmNo
      and
       Reg.s# = @s#
update STDTRM  SET TrmGpa=(@sumGrade/@SumUnit)
     where stdtrm.s# =@S#
        and
         stdtrm.TrmNo=@TrmNo
print 'Grade and TrmGpa Updated'
END
```

مثال 4 : تریگر ی می نویسیم که هر بار که یک دانشجو در درسی ثبت نام می کند ، در صورتی که درس اخذ شده جزء دروس گذرانده وقبول شده دانشجو باشد با دادن پیغام مناسب، از ثبت نام وی جلوگیری نماید .

```
create trigger Tr_RepeatedCourse
on reg
after insert
as
begin
if ((select count(*) from reg,inserted
          where
           REG.c# = inserted.c#
          and
           REG.s# = INSERTED.s#
          and
           REG.TrmNo <> Inserted.TrmNo
          and
           REG.grade> (select passgrade from CRS where CRS.c#
=Inserted.c#))>0)
begin
print 'You register this course in previous terms'
```

```
rollback transaction
return
end
print 'You register this course '
commit transaction
End
```

مثال 5 : می خواهیم تریگری بنویسیم که هر بار که یک دا نشجو در درسی ثبت نام می کند تعداد افرادی را که در آن درس ثبت نام کرده اند محاسبه نموده ، در صورتی که تعداد آن ها بیش از 60 نفر باشد با دادن پیغام مناسب ، ازثبت نام وی جلوگیری نماید.

```
create trigger Tr_CRS_Capacity
ON REG
After  INSERT
AS
Begin
declare  @C# char(7)
declare @cnt int
DECLARE @TrmNO char(4)
select  @C#=C# , @TrmNO=TrmNO  from INSERTED
select  @cnt=count(*) from reg
      where
        REG.c# =@c#
        and
        REG.TrmNo = @TrmNO

if @cnt > 60
begin
print 'This course have not capacity!'
rollback transaction
return
end
commit transaction
End
```

1- با استفاده تابع **TotRegUniFn_Update** تریگری به نام **Tr_Update_TotRegUnit** بنویسید که هر بار که در جدول **REG** رکوردی درج یا حذف می شود مقدار جدید**TotRegUnit** در جدول **STD** اصلاح کند.

2 - تریگری به نام **Tr_Update_Gpa_TotPassUnit** بنویسید که وقتی نمره دانشجو در جدول **REG** اصلاح می شود معدل کل دانشجو**(Gpa)** و مجموع واحد پاس شده دانشجو **(TotPassUnit)** در جدول **STD** اصلاح شود .

3- تریگر به نام **Tr_Del_Prereq** بنویسید که هر بار که یک دا نشجو در درسی ثبت نام می کند قبل از درج ثبت نام بررسی کرده در صورتی که درس های پیش نیاز آن را اخذ نکرده باشد با دادن پیغام مناسب، از بت نام وی جلوگیری نماید**.**

4- یک تریگر به نام **Tr_Unit_Limit**بنویسید که هر بار که یک دا نشجو در درسی ثبت نام کند و یا ثبت نام خود را از نظر درس اخذ شده اصلاح کند . مجموع وا حد های ثبت نامی اورا در ترم جاری محاسبه نموده ، در صورتی که بیش از20واحد ثبت نام نموده بعد از دادن پیغام مناسب ، ازثبت نام وی جلوگیری نماید .

آزمایش 6

پیش آگاهی

در این آزمایش می خواهیم جدول جدیدی تعریف کنیم وبااستفاده از آن به سؤالاتی پاسخ دهیم که در آنها ارجا ع جدول به خود جدول صورت می گیرد که دراین صورت برای **join** کردن جدولی باخودش الزاماباایدازنام مستعاراستفاده شود .

جدول زیررادرنظرمی گیریم :

EMPLOYEE(E#,Ename,EMgr#)

دراین جدول، **E#** شماره کارمندیبانوع داده ای **nchar(5)** ، **Ename** نام کارمند بانوع داده ای **nvarchar(50)** و**EMgr#** شماره کارمندی رئیس کارمندبانوع داده ای**nchar(5)** می باشد و کلید اصلی فایل ،صفت خاصه **E#** می باشد و**EMgr#** راکلیدخارجی که به **E#** رجوع می کند ، تعریف می کنیم .

مثال : می خواهیم تابعی به نام **fn_FindReports** تعریف کنیم که مقدارپارامتر**INE#** رابه عنوان ورودی دریافت کرده ویک جدول بانام **retFindReports** ،شامل شماره ونام کارمندان زیردست کارمند**INE#** رادرکلیه سطوح تاپایین ترین سطح رابرگرداند . توجه شودکه الگوریتم مذکوربه صورت کلی یک الگوریتم بازگشتی می باشد که به دلیل عدم وجود امکانات بازگشتی در**MS_SQL** الگوریتم مذکورشبیه سازی شده است .

```
Create function fn_FindReports (@INE# nchar(5))
RETURNS @retFindReports TABLE (E# nchar(5) primary key,
  Ename nvarchar(50) NOT NULL,
  EMgr# nchar(5)  refrences EMPLOYEE(E#))
AS
BEGIN
```

متغیری رابرای نگاه داشتن تعداد سطرهای اضافه شده به جدول کمکی ، تعریف می کنیم .

```
DECLARE @RowsAdded int
```

جدولی تعریف می کنیم تا نتایج میانی تابع رادرآن نگاه داریم ودراین جدول یک فیلد به نام **PROCESSED** تعریف می کنیم تابدین وسیله بتوانیم سطرهای پردازش شده راازسطرهای جدیدواردشده، جداکنیم .

```
  DECLARE @reports TABLE (E# nchar(5) primary key,
    Ename nvarchar(50) NOT NULL,
    EMgr# nchar(5),
    processed tinyint default 0)
```

جدول ایجادشده رابادادن مشخصات مربوط به کارمندی که شماره مستخدمی آن به عنوان ورودی داده شده ، توسط جدول مقداردهی اولیه می کنیم . **EMPLOYEE**

```
 INSERT @reports
  SELECT E#, Ename, EMgr#,0
  FROM employee
  WHERE E# = @InE#
```

@RowsAdded درابتدابامقدار1 **set** می شود.

```
SET @RowsAdded = @@rowcount
```

تاوقتی که **@reports** اضافه شده که زیردست هایش پیدانشده، کارهای زیررانجام بده .

کارمندجدیدی در جدول

```
  WHILE @RowsAdded > 0
  BEGIN
```

فیلد**processed** هرکارمندی که قراراست زیردست هایش پیداشود رابامقدار1 **set** کن .

```
    UPDATE @reports
    SET processed = 1
    WHERE processed = 0
```

مشخصات زیردست های هرکارمندکه **processed** برابر 1دارد رادرجدول **reports** وارد کن .

```
    INSERT @reports
```

```
    SELECT e.E#, e.Ename, e.EMgr#, 0
    FROM employee e, @reports r
    WHERE e.EMgr#=r.E# and e.EMgr# <> e.E# and r.processed = 1
```

تعدادسطرهایی که جدیدا به جدول **@reports** اضافه شده رادرمتغیر **@RowAdded** بریز.

```
SET @RowsAdded = @@rowcount
```

فیلد**processed** کارمندانی که زیردستان آن ها پیداشده رابامقدار2 **set** کن

```
    UPDATE @reports
    SET processed = 2
    WHERE processed = 1
  END
```

نتیجه نهایی رادرجدول خروجی تابع وارد کن

```
  INSERT @retFindReports
  SELECT E#, Ename, EMgr#
  FROM @reports
  RETURN
END
GO
```

دستورکار

1- جدول **EMPLOYEE** رابامشخصات گفته شده درقسمت پیش مطالعه ایجاد کرده وآن راباداده مناسب پرنمایید . وسپس به سؤالات زیرپاسخ دهید .
2- تابع مثال ارائه شده درقسمت پیش آگاهی را به کمک دستور **select** وباشماره کارمندی های مختلف ، احضار نموده ونتایج حاصله رابررسی نمایید .
3- یک تابع بنویسید که شماره کارمندی یک کارمند راگرفته ونام رئیس ونام رئیس رئیس کارمند رااستخراج نماید .
4- تابعی بنویسید که شماره کارمندی یک کارمند راگرفته ونام کارمندانی رااستخراج کند که رئیس آن ها رئیس کارمندمذکورباشد .(یعنی کارمندانی که رئیس مشترک دارند .)
5- یک تابع بنویسید که شماره کارمندی کارمند را به عنوان ورودی گرفته ونام زیردستان کارمندرادرسطح چهارم استخراج نماید.
6- تابعی تعریف کنید که شماره کارمندی یک کارمند راگرفته ونام رئیس کارمندرادرسطح چهارم استخراج کند .
7- یک تابع بنویسید که شماره کارمندی یک کارمند راگرفته ونام کلیه رؤسای کارمندمذکوررااستخراج نماید .

آزمایش پنجم

معرفی ساختارهای از زبان **C#**که در این آزمایشگاه از آن ها استفاده شده است.

پیش مطالعه:

با توجه به آشنایی دانشجویان این درس، با زبان **C** در این بحث تنها به معرفی مختصر ساختارهایی که در پیاده سازی بخش " برنامه کاربردی " این آزمایشگاه کاربرد دارند از زبان **C#** می پردازیم.

بخش 1

1- تعریف متغیرها : تعریف متغیرها در زبان **C#** مانند زبان **C** صورت می گیرد مثال :

Float
m,n

2- تعریف ثابت های نمادین : ثوابت در زبان **C#** نیز مانند زبان **C** با استفاده از **#define** تعریف می شوند .مثال:

#define
Myvalue 100

3- انواع داده ای : برخی از انواع داده ای پرکاربرد در **C#** عبارتند از : **double ,bool,float,int,char** محدوده کاربرد این انواع داده ای عیناً مانند زبان **C** می باشد. در**C#** پیش وند های**Long , unsigned , Signed** و **Short** همراه با بعضی از انواع داده ای بالاقابل استفاده هستند .مثلاً نوع داده ای **char** در حالت عادی محدوده 127- تا 127 را شامل می شود ولی **unsigned char** محدوده 0 تا 256 را در بر می گیرد یا مثلاً متغیر **Long int** محدوده

21 47 48 36 47 – تا 21 47 48 36 47 را می پوشاند. از بررسی محدوده بقیه متغیرها به علت نداشتن کاربرد در این آزمایشگاه چشم می پوشیم.

4- عملگر های زبان **C#**مانند زبان **C**عبارتند از:

- محاسباتی:**- -,%,/,*,-,+,=**و++
- رابطه ای:**=!,==,>=,<=,>,<**
- منطقی:**|| ,&&,!**

در زبان **C#** می توان از عملگرهای ترکیبی = +,= –,= *,= / ,= % مانند زبان **C** استفاده کرد.

بخش دوم: آرایه ها در **C#**

- آرایه های یک بعدی :

برای تعریف آرایه ها در زبان **C#** از ساختار زیر استفاده می کنیم.

DataType [] Arrayname = new Aray type [No of elements]:

No of elementsمی تواند توسط یک متغیر هم تعیین شود.

مثال **:**

Int[] b=new int [L] یا **Floot [] a1= new Floot[5]**

- آرایه های چند بعدی

برای تعریف آرایه های چند بعدی نیز از ساختاری مشابه استفاده می کنیم.

برای مقدار دهی اولیه به یک آرایه در هنگام تعریف کافی است مقادیر اولیه را در {} در جلوی تعریف آرایه قرار دهید

برای مقدرا دهی اولیه به آرایه های چند بعدی می توانید از روش زیر هم استفاده کنید:

Int [,] a=new int [2,3]{1,3,5,7,9,11}

تعدادعناصرآرایه : تمامی آرایه از کلاسی به نام **Array** ارث می برند. این کلاس خاصیتی به نام **Length** دارد که از آن می توانید برای تعیین تعدادعناصرآرایه استفاده کنید.

مثال:

Int l=a.Length;

بعد از این در**l** مقدار 6 قرار می گیرد.

در این جا این نکته را هم ذکر می کنیم که کلاس **Array** 10 متد مختلف را هم پیاده کرده است. که معمولاً با دستورات عادی قابل پیاده سازی هستند یکی از پرکاربردترین این متدها ، متد **Sort** است که عناصر آرایه را مرتب می کند و نحوه کاربرد آن به شکل زیر است.

Array . Sort (Array name):

مثال:

Array.Sort (a1):

این نکته را نیز یاد آور می شویم که پارامتر این تابع باید یک آرایه تک بعدی باشد.

بخش سوم :

شرط هاو حلقه سازی در **C#** :

ساختار های شرط وحلقه سازی در زبان **C#**از جمله **if,switch,for,while,do…while**درزبان **C#**عینا مانند زبان **C**قابل استفاده هستند.

بخش چهارم:

کلاس ها، متدها و خصوصیات آنها در **C#** :

نوشتن متدهای مربوط به کلاس ها:

برای نوشتن متدها باید نوع، نام ، پارامترها و بدنه تابع مشخص شود.

```
Private  int getarg(int a,int b)
{
Int temp;
Temp (a+b)/2;
 Return (temp);
}
```

-نوشتن متدهای همنام (**overloaded**) : توابع یک کلاس می توانند همنام باشند ولی لازم است نوع یا تعداد

پارامتر های آن ها جهت شناسایی در زمان احضار متفاوت باشدیعنی این توابع پارامتر های متفاوتی (چه ازنظر نوع چه از نظر) تعداد دریافت کنند در این صورت با توجه به پارامترهای هایی که در زمان اجرا استفاده می شود به یکی از آن توابع ارجاع می شود مثال :

```
int findMax(int a,int b)
{
  If(a>b)
    Return(a);
  else
    Return(b);
}

int finaMax(float a, float b)
{
بدنه این تابع هم مشابه تابع قبلی است.
}
```

حال فرض کنید درزمان اجرا **find max** صدا زده شود در این صورت در صورتی که هر دو پارامتر آن از نوع **int** بودند مقدار بازگشتی از نوع **int** و در صورتی که هر دو پارامتر آن **float** بودند مقدار بازگشتی از نوع **float** است.

جنبه دیگری از نوشتن متدهای همنام نوشتن متدهای یکسان (با پارامترهای یکسان) در 2 کلاس است در این صورت این توابع با توجه به نام کلاسشان که قبل از نام متد می آید مشخص می شوند.این جنبه از نوشتن توابع همنام را چند ریختی می نامیم.

-نحوه ارسال آرگومان ها برای توابع: برای ارسال آرگومان ها به توابع عیناً مانند زبان C عمل می شود در این صورت ارسال پارامترها **by value** خواهد بود یعنی تغییر پارامتر ها در تابع هیچ تاثیری در آرگومان های احضار نخواهد داشت .ولی اگر بخواهیم فراخوانی **by refernce** باشد در این صورت یکی از روش های ممکن استفاده از کلمه کلیدی **Out** است: مثال :

```
Private void Inc (out int x,out int y)
{
X++;
Y++;
}
```

در صورتی که ازکلمه کلیدی**out**استفاده شود حتی می توان پارامتر هایی که از قبل مقدار دهی نشده اند را به این توابع فرستاد ومثلا در ابتدای تابع آن ها را مقدار دهی کردیعنی می توان از آن ها به عنوان خروجی تابع استفاده کرد.

مثال:

```
Private void Initialize(out int x,out inty)
{
X=5 ;
 Y=8;
}
```

در هنگام ارسال پارامتر به متدهایی که پارامترهایی با مشخصه **out** دارند باید یک کلمه کلیدی **out** را نیز ذکر کرد.

مثلاً

```
Int x,y;
Initialize (out x,out y)
```

نوشتن متدهایی با پارامترهای از نوع آرایه :

نوشتن توابعی که پارامترهای از نوع آرایه دارند هیچ تفاوتی با نوشتن توابع معمولی ندارد. حتی این امکان وجود دارد که قبل از نام تابع کلمه **out** را ذکر می کرد در این صورت ارسال آرایه **by reference** خواهد بود.

مثال فرض کنید که تابعی به شکل کلی زیر نوشته ایم که وظیفه آن مرتب کردن آرایه **a** در محدوده بین**Fpoint,Spoint** و برگرداندن مقدار بزرگترین عنصر آرایه در این محدوده است**.**

```
Private int Myfunc (int[] a, int Spoint , int Fpoint)
{
....
}
```

. این تابع را مثلاً می توان به شکل زیر استفاده کرد.

```
Int Max,
int [ ] Array = new Array [8] :{8,4,4,14,13,2,1,0}
Max=My func(Array ,1,5)
```

- نحوه ارث بری یک کلاس از کلاس دیگر: در زبان **C#** نیز مانند زبان های دیگر شی گرا برای این که یک کلاس از کلاس دیگر ارث ببرد. از عملگر : استفاده می شود

فرض کنید کلاسی به نام **Shape** تعریف کرده ایم در این صورت کلاس **Triangle** می تواند از آن ارث ببرد و خصوصیات ومتدهای مورد نیاز خود را به آن اضافه کند یا حتی خصوصیات و متدهای کلاس **Shape** را باز نویسی کند.

در صورت ارث بردن یک کلاس از کلاس دیگر تمامی متدها وخصوصیات **Proteted , Public** کلاس پایه به کلاس فرزند ارث می رسد مثلا اگر کلاس **Shape** خصوصیتی به نام **xleftouterpoint** داشته باشد این خصوصیت برای کلاس **Triangle** هم قابل استفاده خواهد بود. : مثلاً

```
Triangle MyTriangle= new Triangle()
                    ;1MyTriangle .xleftouterpoint =
```

استفاده از کلمه کلیدی **new** وبعدآوردن نام کلاس موجب ایجاد یک شی جدید از یک کلاس می شود. و در صورتی که کلاس سازنده ای داشته باشد آن سازنده هم صدا زده خواهد شد. در تعریف بالا بعد از تعریف شی **My MyTriangle** ازکلاس **Triangle** مقدار خصوصیت **xleftouterpoint** آن به1 ،**Set** شده است.

معرفی کنترل های فرمی مورد نیاز در آزمایشگاه از زبان **C#**:

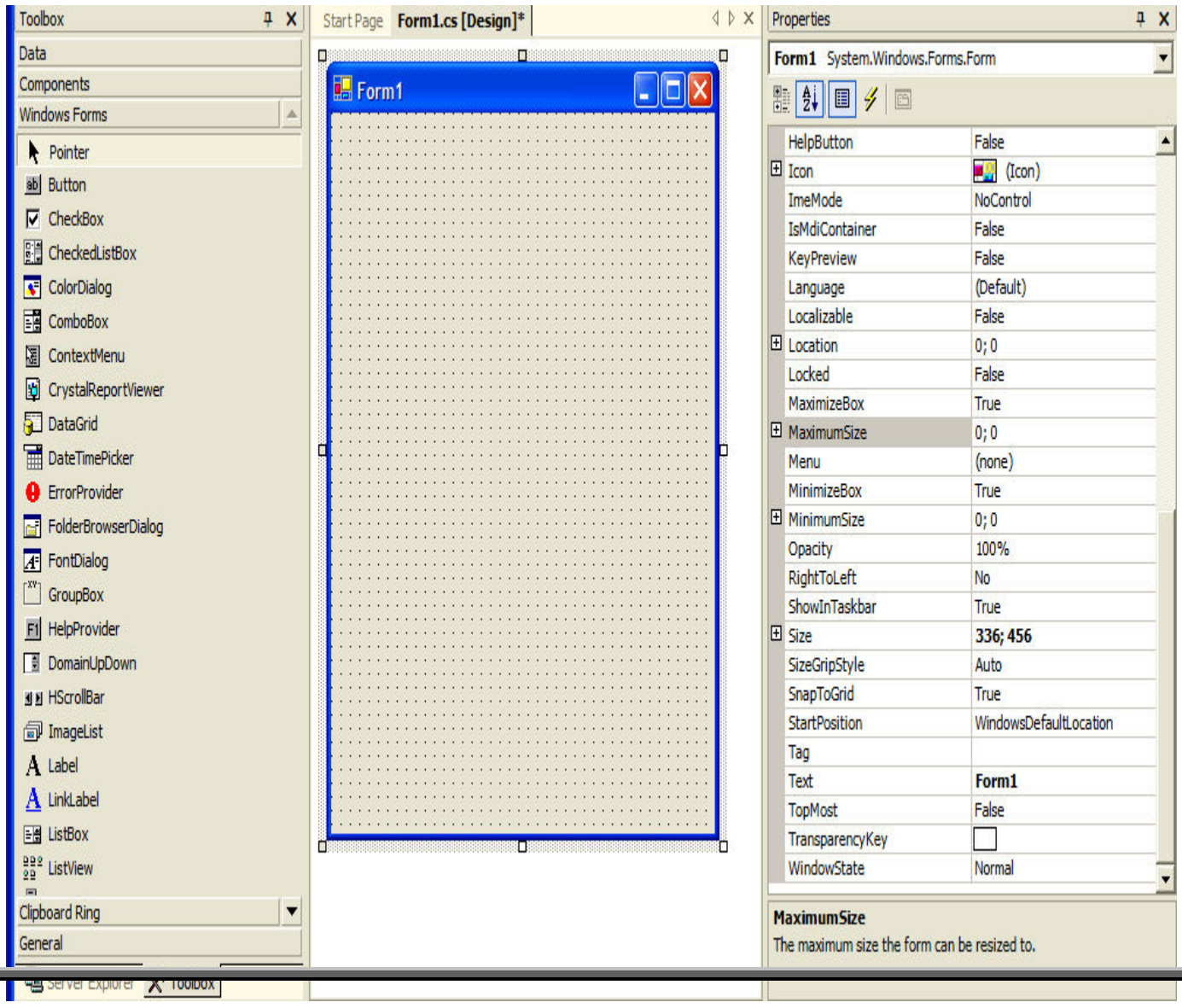
محیط **C# Visual Designer** :

پنجره تنظیم رویدادها

وخصوصیات

جعبه ابزار

پنجره **Form Designer**



برای ایجاد فرم های تحت ویندوز کافی است :

1-کنترل های مورد نیاز خود را به فرم موجود اضافه کنید.

2- با استفاده از پنجره خصوصیات(**properties**)صفات کنترل را با مقادیر دلخواه خودتان تنظیم کنیدیا تنظیم خصوصیات آن به مقادیر دلخواه خودرا کدنویسی کنید.(مثلا در قبال کلیک کردن یک کلیدخصوصیت رنگ فرم به مقداری که انتخاب شده تنظیم گردد.)

2-اعمالی که بایددر قبال رویدادهایی که روی کنترل ها اتفاق می افتد انجام شود را کدنویسی کنید.مثلا در صورت کلیک کردن بر روی یک کلید فرم دیگری لود شودیا به عبارتی متد لود کننده فرم دوم احضار گردد.

بنابراین برای ایجاد فرم ها ی تحت ویندوز نیاز به شناخت خصوصیات خود فرم و کنترل های موجود در جعبه ابزار **C# Visual Designer**داشته و برای شناخت این کنترل هانیاز به شناخت خصوصیات،متدهاورویدادهای تعریف شده روی این کنترل هاست .در این قسمت به معرفی برخی خصوصیات،متدهاورویدادهای تعریف شده روی کنترل های مورد نیاز در آزمایشگاه می پردازیم .ازآن جا که بسیاری از این خصوصیات یا متد ها در بین کنترل های مختلف وخود فرم مشترکند از تکرار آن ها در همه کنترل ها چشم می پوشیم و فقط در موارد اولیه آن ها را ذکر می کنیم. بسیاری از خواصی که اثرآن ها باتغییردر **C# Visual Designer**قابل مشاهده می باشد نیز ذکر نخواهد شد.

1-شی فرم:

1-1خاصیت ها :

- **Name**:نام فرم است. دو فرم هم نام دریک پروژه **ont**:فونت پیش فرض به کار رفته در کنترل هایی که به فرم اضافه می شوند را نشان می دهد.
- **AutoScrollMargins**:با این خاصیت می توان اندازه حاشیه های نوار جابجایی را تعیین کرد .این خاصیت خود شامل دو خاصیت **Width**و**Height**است.

- **Autoscroll**:تعیین می کند آیا اصلا نوار جابجایی به فرم اضافه شود یا نه.
- **Location**:مکان گوشه سمت چپ وبالای فرم در هر لحظه را تعیین می کند .وخود شامل 2 خاصیت **X**و**Y**است.
- **StartPosition**:محل قرار گرفتن فرم را در هنگام لودشدن مشخص می کند.
- **Enable**:تعیین می کند که اطلاعات فرستاده شده به فرم پردازش شود یا خیر. اگر این خاصیت**False**باشد فرم به هیچ رویدادی پاسخ نمی دهد.
- **CancelButton**:نمایانگرنام کلیدی است که در صورتی که کاربر کلید **Esc**را روی فرم فشا ردهد معادل با فشار دادن آن کلید است و در نتیجه مجموعه دستورات معادل با رخ دادن رویداد کلیک روی آن کلید اجرا می شود.
- **Opacity**:میزان شفافیت فرم را تعیین می کند.
- **RighttoLeft**:در صورتی که این خاصیت **true**باشد اطلاعات از راست به چپ نمایش داده می شوند.
- **Language**:زبان مورد استفاده در فرم را تعیین می کند.

2-1رویدادهای قابل رخ دادن درون یک فرم :

- **Activated**:این رویداد با فعال شده فرم روی می دهد.
- **Click**:با کلیک کردن روی فرم رخ می دهد.
- **DoubleClick**:با کلیک مضاعف روی فرم رخ می دهد.
- **Closed**:با بسته شدن فرم رخ می دهد.
- **KeyPress**:با فشردن هر کلید صفحه کلید این رویداد رخ می دهد.رویداد **KeyPress**به صورت زیر **handle**می شود:

```
Privare void نام کنترل_KeyPress(object
sender,System,Windows.Forms.KeyPress EventArgs e)
{
}
```

روشن است که پارامتر **sender**نمایانگرشی اطلاع دهنده این رویداد است.یکی از خواص پرکاربرد ساختار**e**نیز **KeyChar**است که حاوی کاراکتر فشرده شده است. که می توان در **handle**کردن رویداد از آن استفاده کرد.

- **Load**در هنگام لود شدن فرم رخ می دهد.

3-1متدها

- **Activate**:برای فعال کردن فرم به کار می رود.نحوه استفاده از آن در حالت برنا مه نویسی این گونه است:

نام فرم.**ActiveForm.Activate**()

ActiveFormفرم در حال اجرای برنامه جاری ر ا تعیین می کند.

- **Close**:منابع گرفته شده توسط فرم را آزاد می کند و فرم را می بندد. نحوه استفاده از آن در حالت برنا مه نویسی این گونه است:

نام فرم.**ActiveForm.Close();**

- **Hide()**:فرم را مخفی می کند.
- **Show()**:کنترلی را که مخفی بوده است را نمایش می دهد.
- **Focus()**:کنترل جاری را به فرم احضار کننده این متد می دهد.
- **Refresh()**:محتویات فرم را **Refresh**می کند.

2-شی **textbox**

کاربرد این شی معمولا در گرفتن یک رشته از کاربر یا نمایش یک رشته به وی است.

2-1خاصیت ها :

- **MaxLength**:حداکثر طول متنی که کابر می تواند وار دکند را تعیین می کند.
- **MultiLine**:قابلیت این که جعبه متن چندین سطر متن در یافت کند را تعیین می کند.
- **ScrollBars**:می توان تعیین کر دکه آیا جعبه متن نوار جابجایی داشته با شد یا نـه و نـوع آن را تعیین کرد.
- **ReadOnly**:تعیین کننده این خاصیت است که آیا میتوان مـتن درون **textBox**راتغییـر داد یا نه.
- **Anchor**:نشان دهنده مکان نمایش متن جعبه متن، درون جعبه متن است وخواص خـود را از **AnchorStyales**می گیرد.مثلا اگر بخواهیم در حالـت برنامـه نویسـی بـه ازای حـادث شدن رویداد خاصی مکان نمایش را به بالای **textBox**منتقل کنیم کد زیر را می نویسیم :

textBox1.Anchor=AnchorStyles.Top;

2-2متدها:

- **AppendText**:این متد رشته ای را به عنوان پارامتر در یا فت و آن را به انتهای مـتن جعبه متن اضافه می کند.

3-شی**Label**:

کلیه خواص،متدهاورویدادهای دارای کاربرد این شی کنترلی در موارد قبلی توضیح داده شد .

4-شی **Button**:

کلیه خواص،متدهاورویدادهای دارای کاربرد این شی کنترلی در موارد قبلی توضیح داده شد .

مثال:در این جا به عنوان تمرین یکی از فرم های پروژه ثبت نام را تولید کنیم.

مثال:فرض کنید می خواهیم یکی از فرم های این پروژه کـه بـرای درج نـام یـک دانشـجوی جدیـد یـا نمایش اطلاعات یک دانشجوی خاص به کار می رود را تولید کنیم.شکل نهایی فرم شکل3است.

برای ایجاد این فرم قدم های زیر را طی کنید:

1-پروژه جدیدی از نوع **Windows Form**باعنوان **registration**ایجاد کنید.

2-خاصیت **Text**اشی **Form1**را به "فرم اطلاعات دانشجو" تغییردهید.

3-خاصیت **RighttoLeft**را به **yes**تنظیم کنید.

4-می توانید **font**ر انیز به مقدار دلخواه خود تغییر دهید.

5-**startPoition** را با **CenterScreen**مقداردهی کنید .

6-**Language**را با **farsi**مقدار دهی کنید.

7- **autoScroll**ر ا**true**کنید تا در صورت نیاز نوار جابجایی به فرمتان اضافه شود.

8-کنترل های **label**،**textBox**و**Button**های نشان داده شـده را بـه فـرم اضـافه کنیـدو خـواص **font**،**Righttoleft**و**text**آن ها ر امقدار دهی کنید.خاصیت متن جعبه متن(**textbox**) ها رافعلا با تهی مقدار بدهید.خاصیت **ReadOnly** مربوط به **Label**های معدل دانشجو، مجموع واحد هـای پاس شده دانشجو، مجموع واحد های ثبت نامی دانشجو را با **true**مقدار دهی کنیـد(چـرا؟)(راهنمـایی :این فیلد ها فیلد هایی هستند که از فیلد های دیگر پایگاه داده محاسبه می شوند.)

9-خاصیت **Name**جعبه متن های موجود را بانام فیلد های متناظرشان در جدول **STD**مقدار دهـی کنید(به جای **#s**از **sNo**اسـتفاده کنیـد) مثلاجعبـه متنـی کـه در جلـوی نـا م دانشـجو قـرار دارد را **Name**نام گذاری کنید.این کار در کدنویسی ارتباط دادن فیلد های بانک اطلاعاتی وجعبه متن هـا را راحت تر می کند.

10-چهـــــــــار کلیـــــــــد موجـــــــــود را بـــــــــه ترتیـــــــــب **btnAdd**و**btnDelete**و**btnUpdate**و**btnSearch**نامگـذاری کنیـد.(ایـن روش یـک روش استاندارد است)

حال به معرفی بقیه کنترل های مورد نیاز مان می پردازیم:

5-شی**checkBox**:

این شی دو حالت فعال و غیر فعال را به خود می گیرد.

5-1خاصیت ها :

- **Checked**:**true**بودن این خاصیت به معنای فعال بودن **checkBox**است.

2-2رویدادها:

- **Checkedchanged**:وقتی خاصیت **checked**تغییر کند این رویداد رخ می دهد.

6-شی**radioButton**:

این کنترل نیز خواص **checkBox**ها را دارد.مهم ترین کاربرد آن ها به وجود آوردن حالت های مختلفی از تایید شدن یا نشد ن گزینه هاست.

7-شی**groupBox**:

برای کنار هم گذاشتن چند کنترل(مثلا چند **radioButton**)به کار می رود.در این صورت تنها اجازه انتخاب یکی از این کنترل ها به کاربر داده می شود.

8-شی **listBox**:

این کنترل برای نگهداری لیستی از اشیاء مثلا نام دانشجویان کاربرد دارد.

8-1خاصیت ها :

- **Items**:با استفاده از این خاصیت می توان لیست گزینه هایی که در **listBox**باید ظاهر شوند را تعیین کرد.
- **MultiColumn**:تعیین می کند که کنترل می تواند چند ستون داشته باشد یا نه.
- **Datasource**:در صورت لزوم نام منبع داده ای که می خواهیم گزینه های **listBox**از آن تعیین شود را تعیین می کند.منابع داده ای را در آزمایش بعدی مفصلا بحث خواهیم کرد.
- **Sorted**:تعیین می کند که گزینه های **ListBox**مرتب شده باشد یا خیر.
- **SelectedIndex**:اندیس گزینه ای که باید انتخاب شود را تعیین می کند.این اندیس گذاری از صفر شروع می شود.
- **SelectedValue**:مقداری را تعیین می کند که گزینه مربوط به آن باید انتخاب شود.

8-2رویدادها :

- **DatasourceChanged**:این رویداد در صورت تغییر **Datasource**روی می دهد.
- رویدادهای **SelectedIndexChanged**و**SelectedValueChanged**:به ترتیب هنگامی رخ می دهند که خواص **SelectedIndex**و**SelectedValue** تغییر کند.

8-3متدها :

- **Clear**:تمام گزینه های کنترل را حذف می کند.به عنوان مثال دستور زیر تمامی گزینه های **myListBox**را حذف می کند.

```
myListBox.items.Clear();
```

- **Add**:برای اضافه کردن یک گزینه به انتهای لیست عناصر **listBox**به کار می رود

```
citiesListBox.items.Add("بروجرد");
```

- **Insert**:مشابه **Add**عمل می کند با این تفاوت که دو پارامتر دریافت می کند و پارامتر دوم را در موقعیت تعیین شده بوسیله پارامتر اول درج می کندوعناصر بعدی را به سمت پایین شیفت می دهد.

citiesListBox.items.Insert(3,"آبادان ");

- **Count**:تعداد عناصر **listBox**را بر می گرداند.

Int citiesCount;
citiesCount= citiesListBox.items.Count();

- **Remove**:مقداری را دریافت و گزینه مر بوط به آن مقدار را از **listBox**حذف می کند.
- **RemoveAt**:عملکردی مشابه **Remove**دارد با این تفاوت که به جای مفدار شماره اندیسی را در یافت و گزینه مر بوط به آن را حذف می کند.به عنوا ن مثال 2 دستور صفحه بعد معادلند:

citiesListbox.items.Remove("آبادان");
citiesListbox.items.Removeat (3);

یا

- **IndexOf**:مقداری را در یافت می کند و اندیس آن ر ادر **listBox**برمی گرداند.

Int myIndex;
myIndex=Cities.items.IndexOf("یزد");

9-شی **dateTimePicker**:

این شی بهترین کنترل جهت دریافت ونمایش فرمت های تاریخی است مهمترین خصوصیت ایـن شـی خاصیت **value**است که نمایانگر تاریخ این کنترل است .

تا این جا تمامی کنترل هایی که به آن ها نیاز داریم(جز **DataGrid**که در آزمایش بعدی و بعد از معرفی **datasource**ها بررسی می شود)را بررسی کردیم .احتمالا اگر تا به حال از این کنترل ها استفاده نکرده باشیداین سوال برای شما مطرح است که ارتباط بین متد ها ،رویدادها و صفات چگونه برقرار می شود.

مثال:می خواهیم تغییرات کوچکی در فرم اطلاعات دانشجو بدهیم.

1. **Sex textBox**را حذف کنید و یک **groupBox**با نام **Sex**را جایگزین آن کنیدوفیلد **text**آن را با تهی مقدار دهی کنید.
2. داخل این **groupBox**دو **radioButton**بانام های **male**و**female**قرار دهید و **text**آن ها را "مرد"**,**"زن"مقدار دهی کنید.

سوال :در حال حاضر خاصیت **righttoleft**این **groupBox**چه مقداری دارد؟چرا؟(راهنمایی:دقت کنید کنترل های داخل یک فرم خصوصیات مشترک خود را از کلاس فرم به ارث می برند.)

3. **birthday textBox**را حذف کرده آن را با یک **dateTimePicker**جایگزین کنید ونام آن را **birthDate**اختیار کنید.**dropDownAlign** این کنترل را **right**انتخاب کنید.

در این جا فرم را به شکل نهایی آن آماد ه کرده ایم حال برای این که روی رویدادها هم کمی تمرین کرده باشیم بهتر است برخی از رویداده ای این فرم ر ا**Handle**کنیم.قبل از آن چون در قسمت بعدی برای اطمینان از حاصل کارمان از شی **MessageBox**استفاده می کنیم ابتدا به معرفی این شی می پردازیم.

مهم ترین متد این شی متد **()Show**است که یک جعبه پیا م را روی صفحه نمایش نشان می دهد.یکی از روش های صدا زدن این متد به این شکل است که آن را با چهار پارامتر احضار می کنیم که پارامتر اول پیغام مورد نظر را تعیین می کند.پارامتر دوم متن عنوان جعبه پیام ر اتعیین می کند .پارامتر سوم نوع کلید هایی که می خواهیم روس **MessageBox**قرار گیرد را نمایش می دهد .پارامتر چهارم نیز **Icon**مشخص شده روی جعبه پیام را معین می کند.

این تابع مقداری از نوع داده ای **DialogResult**بر می گرداندکه می توانیم با مقایسه مقدار خروجی این تابع با انواع داده ای **DialogResultd**از عکس العمل کار بر در قبال **MessageBox**آگاه شویم.

مثال:

```
DialogResult Key=MessageBox.Show("The source Record have been Modified  Do You want to save?","Save Alert",MessageBoxButtons.YesNo,MessageBoxIcon.Question);
Bool e=(Key==DialogResult.Yes)
```

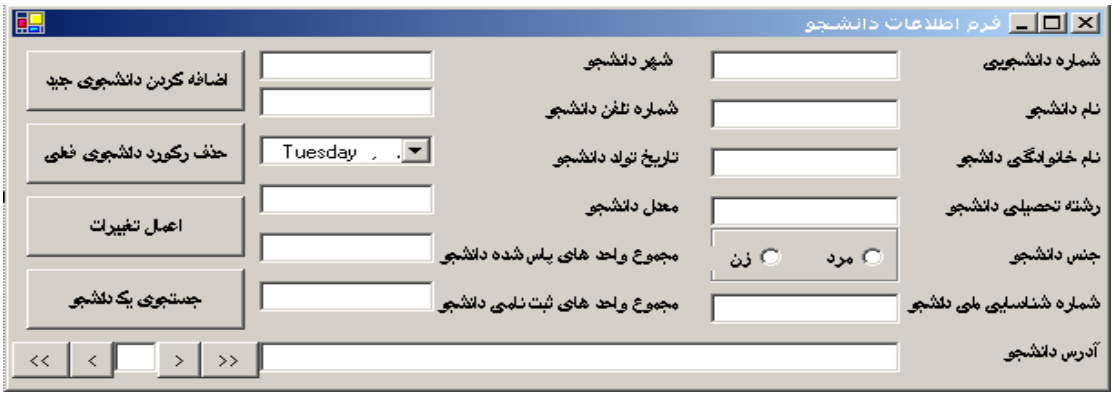
بعد از اجرای کد بالاجعبه پیامی به شکل زیر نمایش داده میشود بعد از انتخاب توسط کاربر، برنامه نویس می تواند با استفاده از متغیر بولین **e**تصمیم گیری کند.

شکل 2

حال می خواهیم رویداد لود شدن "فرم اطلاعات دانشجو"را تعریف کنیم مثلا بخـواهیم بلافاصـله بعـد از لود شدن این فرم رنگ کلید های **btnAdd**و**btnDelate**عوض شودو یـک **MessageBox**هـم لود شدن این فرم را اعلام کند.بنابراین به سراغ بلوک کد **Handle**کننده این فرم می رویم و کدکـد های زیر را جایگزین می کنیم :

```
Private void stdForm_load(Object Sender,System.EventArgs e)
{
   btnAdd.BackColor=System.Drawing.Color.Yellow;
   btnDelete.Backcolor= System.Drawing.Color.Tan;
   MessageBox.Show(" فرم اطلاعات دانشجو لود شده
است","Alert",MessageBox.Buttons.Ok,
MessageBoxIcon.Warning);
}
```

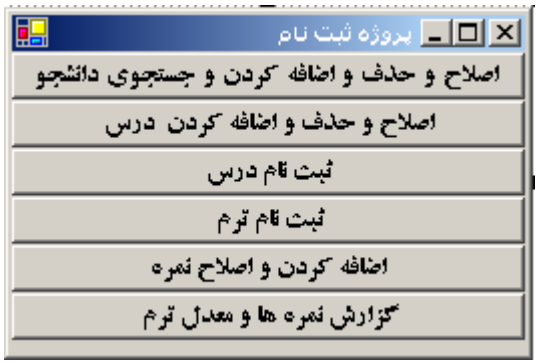
شکل3



دستور کار:

1. پروژه جدیدی از نوع **Windows Application**تعریف کنید.این فرم ،فرم اصلی پروژه با نام **main**خواهد بود.کلید هایی که در فرم نشان داده شده را به آن اضافه کنیدوخصوصیات آن را مانند شکل زیر تنظیم کنید.

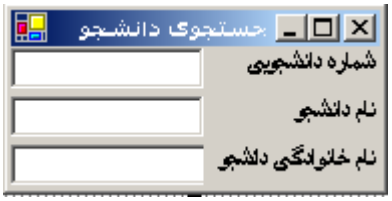
شکل4



از این جا به بعد سعی کنید نام کنترل ها را با نام فیلد های متناظر آن ها در پایگاه داده مشابه بگیرید.

2. فرم **Std**راآن گونه که گفته شدبه پروژه اضافه کنید.

3. فرم زیر را طراحی و به پروژتان اضافه کنید.نام این فرم **Search**خواهد بود.



شکل 5

4. فرمی با نام **Crs** را به شکل زیر طراحی و به پروژتان اضافه کنید .

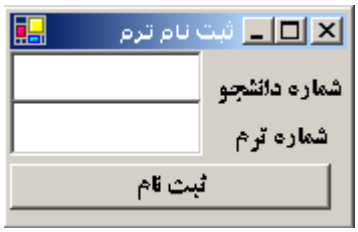
شکل 6

در فرم **Crs**جعبه هایی که با نام **listBox**نشان داده شده اند را از نوع **listBox**بگیرید.برای **listBox**ای که در جلوی **Label**ِ "نوع درس"قرار دارد دو آیتمِ " تئوری" و "عملی"را اضافه کنید.

5. فرمی با نام **coPrAdd**را با طراحی زیر، به پروژتان اضافه کنید:

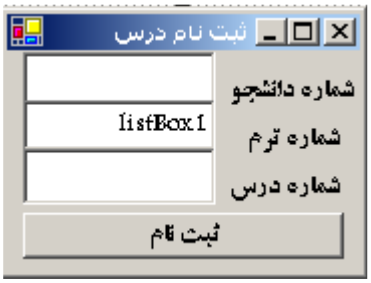
شکل 7

6. فرمی با طراحی زیر و با نام **trmRegistration**را به پروژتان اضافه کنید.



شکل 8

7. فرم زیر را به پروژه ثبت نام اضافه کنید.نام این فرم **crsRegistration**خواهد بود.



شکل 9

دقت کنید که در این فرم جعبه جلوی **label** ِ"شماره ترم" از نوع **listBox**است.

8. اولین فرم صفحه بعد را با نام **courseMarking**به پروژه اضافه کنید.

شکل10

کنترلی که در وسط فرم قرار دارد یک **dataGrid**است که در جلسه بعدآن را پر خواهیم کرد.

9. آخرین فرم پروژه را با نام **trmCourses**به فرم اضافه کنید.این فرم در شکل 11 نشان داده شده است.

شـکل 11

آزمایش های 8و9و10

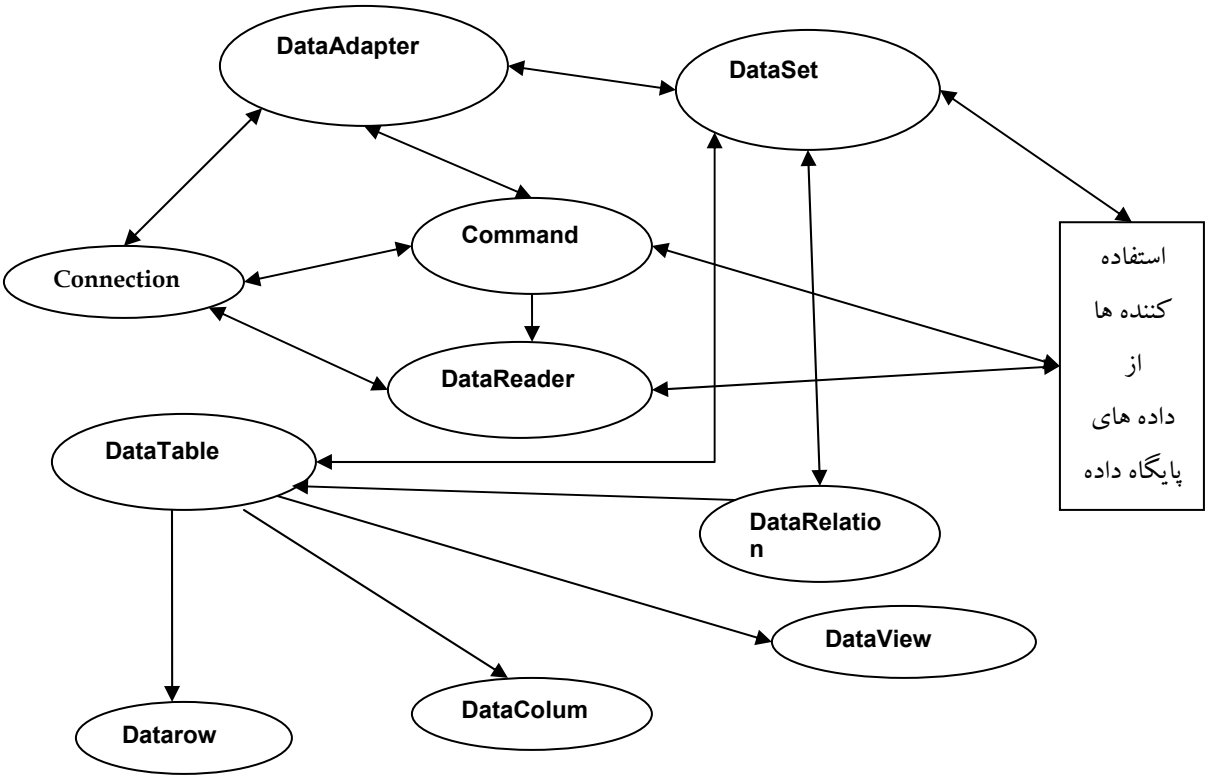
معرفی **ADO.NET**ونحوه استفاده از کلاس های آن درزبان **C#**

پیش آگاهی

مسلما مانند امکانات(کامپوننت های) دیگرزبان **C#**این بخش نیز از کلاس هایی تشکیل شد ه است .در این قسمت به معرفی مختصر این کلاس ها پرداخته و ارتباط آن ها را شرح می دهیم وآن ها را در قالب مثال هایی به صورت کامل تشریح می کنیم.دراین آزمایش تنها به معرف ی کلاس هایی که برای اتصال و استفاده از **SQL-Server** به کار می روند

می پردازیم .کلاس هایی که برای اتصال با بقیه پایگاه های داده ای تدارک دیده شده اند نیز به نحو مشابهی استفاده می شوند.مطلب دیگری که ذکر آن ضروری به نظر می رسد این است که اکثر اعمالی که برای استفاده از یک پایگاه داده باید انجام شوند از دو روش قابل پیاده سازی است روش اول کد نویسی بوده و روش دوم استفاده از ابزارهایی که محیط **.Net Enviroment** برای خودکار کردن این اعمال در اختیار برنامه نویس قرار می دهد.در این آزمایش بنا برنوع عمل مطرح شده یک یا هر دوی این روش ها را ذکر می کنیم.

برای این که مطالبی که بعد از این ذکر خواهد شدرا ساده تردرک شوند کلاس هایی که **ADO.NET**برای کار با پایگاه داده ها در اختیار برنامه نویس قرار می دهدونیز ارتباطات آن ها در شکل زیر نمایش داده شده اند.



معرفی شی**Connection**:

این شی برای برقراری اتصال با پایگاه داده استفاده می شود.این شی از یک رشته(**string**) برای اتصال با پایگاه داده استفاده می کند.دو متد پر کاربرد آن نیز **()open**و**()close**هستند که برای باز و بسته کردن اتصال به کار می روند.

اولین قدم در استفاده از یک پایگاه داده ایجاد ارتباط با آن پایگاه داده ودر حین این عمل تعیین **DataProvider**برای این ارتباط است.**DataProvider**ی که برای ارتباط با پایگاه داده های از نوع **SQL-Server**درنظر گرفته شده است در نامکده **System.Data.SqlClient**قرار دارد.پس اولین قدم برای ایجاد **Connection**اضافه کردن این دو نامکده با استفاده از عبارات زیربه برنامه است.

using System.Data;

using System.Data.Sqlclient;

تعریف **Connection**:

فرم اطلاعات دانشجو(**std**) را لود کنید.در جعبه **ToolBox**گزینه(تب) **Data**راانتخاب کنید.و از این گزینه یک شی **Connection** رابه فرم خود اضافه کنید.برای تنظیم خصوصیات این **Connection**به سراغ پنجره **Properties**برویدوخصوصیت **ConnectionString**آن را به روش زیر تنظیم کنید.

Newconnectionرا انتخاب کنید و فیلد های پنجره **DataLinkProperties**را با توجه به اتصالی که می خواهید داشته باشید تنظیم کنید.در این جا پایگاه داده **registration**راانتخاب کردهبعد از تست این **Connection** ، به کمک گزینه **Test** **Connection**،**DataLinkProperties** را تایید کنید .نام این اتصال به صورت پیش فرض**sqlConnection1**بوده و قابل تغییر می باشد.خصوصیت **ConnectionString**آن با عبارتی مشابه عبارت زیر مقدار دهی شده است:

workstation id=(DBLAB);packet size=4096;integrated security=SSPI;initial catalog=registration;persist security info=False

با این موضوع که این مقدار چگونه بدست می آید کاری نداریم و هر بار برای اتصال حتی در مد کد نویسی از همین روش میتوانید استفاده کنید و مثلا بعد از بدست آوردن این مقدار بااستفاده از کپی کردن همین مقدارواستفاده ازآن در برنامه یتان می توانید خصیصه ی **ConnectionString**مربوط به **Connection**رامقدار دهی کنید.حال کافی است این **Connection**را باز کنیم.برای این کارکافی است مثلادر هنگام لود شدن فرم(روی دادن رویداد **std_Load**) متد **()open**این **Connection**را با عبارت زیرصدا بزنیم.

```
private void std_Load(object sender, System.EventArgs e)
{
    sqlConnection1.Open();
}
```

می توانستید حتی تعریف این **Connection**را در مد کد نویسی انجام دهید.برای این کار بعد از کپی کردن مقدار **ConnectionString**می توانیدارتباط موجودِ**sqlConnection1** را حذف کنید و قطعه کدهای زیررا جایگزین کنید:

```
private void std_Load(object sender, System.EventArgs e)
```

```
{
   string mySqlConnectionStr="workstation id=(Local);packet
size=4096;integrated        security=SSPI;initial
catalog=registration;persist security info=False";
    SqlConnection mySqlConnection1 = new SqlConnection();
   mySqlConnection1.ConnectionString=mySqlConnectionStr;
      mySqlConnection1.Open();
}
```

دریک **ConnectionString**،**Workstataion id**نام کامپیوتری است که سرویس دهنده مورد نظر ما برای اتصال روی آن قرار دارد.

معرفی اشیاء**DataCommand** و **DataReader**:

1-**DataCommand**:همان گونه که از نامش پیداست این شی برای فرستادن تقاضای اجرای دستورات **Sql** یا رویه های ذخیره شده در پایگاه داده ها به کار می رود.این شی برای اجرای بعضی از دستورات **Sql**از قبیل **Select**، **Insert**، **Delete** و**Update**کردن اطلاعات و احضار رویه ها استفاده می شود.این شی رادر صورت نیاز تنها با کمک شی **Connection**و بدون کمک گرفتن از هر شی دیگری می توان برای اجرای دشتوراتی که از آن ها انتظار خروجی نداریم یا خروجی فقط یک مقدار مشخص می باشد استفاده کرد مانند دستورات بهنگام سازی داده ها،اجرای دستورات**DDL**یا به صورت کلی اجرای دستوراتی که نتیجه برگشتی با بیش از یک سطر یا ستون ندارند.

اما وقتی خروجی یک **DataCommand** ،بیش از یک سطر یا ستون می باشد نیاز به شی **DataReader**حس می شود.شی **DataReader**جریانی فقط خواندنی وفقط رو به جلواز نتیجه یک **DataCommand**را فراهم می کند.در واقع با استفاده از **DataReader**می توان روی نتایجی که **DataCommand**،آن ها را **Fetch**می کندحرکت کرد.پس**DataReader**در واقع به کمک شی **DataCommand**معنای واقعی خود را پیدا می کند(باید جدولی از نتایج وجود داشته باشد تا **DataReader**در بین آن ها حرکت کند)

ایجاد**DataCommand**و**DataReader**:

DataCommandها را می توان در زمان طراحی فرم ها یا در زمان اجرا بوسیله کدهایی که نوشته می شود ایجاد کرد ولی ایجادیک **DataReader**هاتنها در زمان اجرا و با استفاده از متد **ExecuteReader**ازکلاس**DataCommand**صورت می گیرد.توجه شود که شی **DataReader**سازنده ای ندارد.معنی این بحث به صورت خلاصه این است که **DataReader**تنها در زمان کدنویسی مقدار دهی اولیه می شود.

برای اضافه کردن یک **Command** از طریق **Net Enviroment.**می توانید از گزینه(تب) **Data**یک **sqlCommand**به فرم اضافه کنید.سپس باید اگر **Connection**های به ثبت رسیده ای دارید **Connection**مورد نظر خود را به عنوان خصوصیت **Connection**این **SqlCommand**تعریف کنید .سپس خصوصیات**CommandText** و **CommandType**این**sqlCommand**راتنظیم کنید.خصوصیت**CommandType** از یک **sqlCommand**نشان دهنده نوع دستوری است که در **CommandText**وجود دارد.دو مقدار مهمی که این متغیر می گیرد عبارتند از:

1-**Text**:نمایانگر این مطلب است که این**Command** حاوی یک دستور متنی **Sql**است.

2-**StoredProcedure**:نمایانگر این است این**Command**برای احضار یک **StoredProcedure**به کار می رود.

مقدار پیش فرض این متغیر **Text**است.

ولی مااز کدنویسی برای ایجادیک **DataCommand**استفاده می کنیم.فرض کنیم می خواهیم ایجاد **DataCommand** در زمان بارشدن فرم صورت گیرد، بنابراین کد های زیر را جایگزین کنید:

```
private void std_Load(object sender, System.EventArgs e)
{
 string mySqlConnectionStr = "workstation id=(Local);packet
size=4096;integrated  security=SSPI;initial
catalog=registration;persist security info=False";
 string mySqlQuery = "select * from reg where s#=8014681";
 SqlConnection mySqlConnection = new SqlConnection();
 mySqlConnection.ConnectionString=mySqlConnectionStr;
 mySqlConnection.Open();
 SqlCommand mySqlCommand = new SqlCommand();
 mySqlCommand.Connection=mySqlConnection;
 mySqlCommand.CommandText=mySqlQuery;
  SqlDataReader myDataReader1  =
mySqlCommand.ExecuteReader();
}
```

در خط آخر دیده می شود که برای مقدار دهی یک**DataReader** از متد**ExecuteReaderd**یک**Command**استفاده شده است واین شی هیچ سازنده ای ندارد.اما باید برای استفاده از اطلاعات موجود در **DataReader**کد هایی نوشته شود.قبل از نوشتن کد ها

لازم است **DataCommand**و**DataReader**رادر حد استفاده در آزمایشگاه بیشتر بررسی کنیم .

مهم ترین متد های شی **Command**عبارتند از:

1. **()ExecuteNonQuery**:با احضار این متداین شی (**Command**)،دستور موجوددر**Text**خودرا اجرا می کند و تعدادسطر هایی که تحت تاثیر قرارمیگیرند را برمی گرداند.کاربرد این متد وقتی است که از رویه ذخیره شده یا دستور **Sql**اجرا شده انتظار خروجی نداشته باشیم .
2. **()ExecuteReader**:این متد**Commandtext**رااجرا کرده ونتیجه ر ادر یک **Datareader**برمی گرداند.کاربرد آن در دستورات **Sql**یاروی رویه هایی است که چندین سطر را بر می گردانند.
3. **()ExecuteScalar**:**Query**رااجرا کرده اولین ستون از اولین سطر مجموعه نتایج را بر می گرداند.کاربرد آن در اجرای رویه ها یا دستورات **Sql**ای است که تنها یک مقدار را بر می گردانند.

متد های مهم یک **DataReader**در زیر لیست شده اند:

1. **()Close**: **DataReader**را می بندد.
2. **()GetName**:نام یک ستون را بر می گرداند.
3. **()IsDbNull**:نمایانگر این است که آیا این ستون مقدار **Null**در خود دارد یا نه.بر حسب این که مقدار فیلدی که به آن اشاره می شود**Null**یاغیر**Null**باشدیکی از مقادیر **True**یا**False**را برمی گرداند.
4. **()Gettype**:از این متد می توان برای بدست آوردن مقدار یک ستون بر حسب یک نوع داده ای دلخواه استفاده کرد.چند نمونه از ساختار های مهم این متد عبارتند از:**GetString,GetInt32,GetFloat,GetDecimal**
5. **()Read**: **DataReader**را به سطر بعدی از مجموعه نتایج پیش می برد.وقتی **DataReader**باز شد کرسر **DataReader**قبل از اولین سطر است و بنابراین برای رسیدن به سطر اول هم یک فراخوانی متد**()Read**لازم است.
6. **()NextResult**:**DataReader**را به نتیجه بعدی از جواب ها پیش می برد.کاربرد آن وقتی است که دستور**Sql**یا رویه ی اجرا شده چندین مجموعه جواب (مثلابا چندین **select**)برگرداند.

دقت کنید که **DataReader**در هر لحظه تنها روی یک سطر داده ها ایستاده است.

مثال:فرض کنیدجعبه لیستی با نام **myListBox** داریم که می خواهیم آن را با مقادیر نام و نام خانوادگی دانشجویان پر کنیم**.** قطعه کد زیر این کار را انجام می دهد.(این مثال از اشیاء تعریف شده در مثال های قبلی بهره می برد)

```
SqlDataReader myDataReader= mySqlCommand.ExecuteReader();
while( myDataReader.Read())
listBox1.Items.Add(myDataReader.GetString(1)+' '+
myDataReader.GetString(2));
```

دقت کنیدچون اندیس ستون ها از صفر شروع می شود **myDataReader.GetString(1)**برای استخراج ستون نام سطر جاری **DataReader**به کار می رود.(**myDataReader.GetInt32(0)**به شماره دانشجویی اشاره می کند)

دراین جابه مطلب مهمیتوجه می کنیم .تا این جا ما میتوانیم هر نوع درخواستی را اجرا کنیم ،جز این که در خواست ما شامل یک یا چند پارامتر باشد(مثلا درخواست مشخصات دانشجویی که شماره دانشجویی وی در یک **textBox**توسط کار بر داده می شود.حال سوال این جاست که چگونه باید مقدار موجود دراین **textBox**را به عنوان پارامتر در درخواستی که به سرور توسط دستورات **T-Sql**می فرستیم جای داد؟)دو روش برای پاسخ دادن به این سوال وجود دارد.روش اول این است که :ابتدا در خواست مورد نظرمان رابه صورت پارامتری در خاصیت **CommandText**یک **DataCommand**قرار بدهیم.یکی از مهم ترین خواص یک **DataCommand**که ذکر آن را به این جا موکول کردیم خاصیت مجمو عه ای **Parameters**است.این مجمو عه می تواند شامل مجمو عه ای از پارامتر ها باشد که پارامتر هایی که در در خواست ها پاس می شوند درون آن قرار می گیرند.

استفاده از پارامتر ها در درخواست هاخیلی ساده است کافی است ابتدااین پارامتر ها را در درخواستهایی که می فرستید مشخص کنید.حالاباید این پارامتر ها در مجموعه پارامتر های **Command**اضافه کنید.بعد از این مرحله باید پارامتر ها را مقدار دهی کنید.

در این جا به سراغ بررسی اولین عمل یعنی نحوه مشخص کردن پارامتر ها در درخواست می رویم .دستوراتی که از پایگاه داده های از نوع **SQL-Server**(و نه انواع دیگرپایگاه داده)استفاده می کنند برای این کار از پارامتر های نامدار که با @شروع می شوند استفاده می کنند.

مثال:

```
mySqlCommand1.CommandText=”select * from Reg where
Trmno=@myTrmno”;
```

خوب در این جا مشخص کردیم که در این دستور **Trmno** یک پارامتر است که بعدا به وسیله پارامتر ها پاس می شود.

اما برای اضافه کردن پارامتر ها به مجموعه پارامتر های یک **Command**متد های زیر در اختیار برنامه نویس است .این متد ها جزو مجموعه **Parameters**از هر **SqlCommand**هستند.

1. **Add(parameterName,parameterValue)**:یک پارامتر را با مقدار **parameterValue**به انتهای مجموعه پارامتر ها ی **Command**اضافه می کند.
2. **Add(parameterName,parameterType,paremeterSize)**´:پارامتری از نوع **paremeterType**و اندازه**paremeterSize**را به انتهای مجموعه پارامتر ها اضافه می کند.پارامتر سوم در نوع داده های مثل **int**کاربرد دارد که مثلا در این موارد پارامتر سوم چند رقمی بودن عدد را مشخص می کند.
3. **Add(parameterName,parameterType)**´:مشابه مورد قبل است و کاربرد آن در مواردی است که نیازی به پاس کردن پارامتر سوم نیست ونوع پارامتر بر حس نوع تبدیل مشخص می گردد.
4. **Insert(parameter Index,parameterValue)**:پارامتری جدید را که مقدار آن برابر با **parameterValue** است را در موقعیتی برابر با **parameter Index**درج می کند.
5. **RemoveAt(parameterIndex)**:پارامتر موجود در اندیس **parameterIndex**ازمجموعه پارامتر های **Command**را حذف می کند.

مثال:

mySqlCommand.Parameters.Add(“@myTrmno”,System.Data.SqlDBType.char,4);

دردستورفوق**@myTrmno**رابانوع داده ای تعریف شده به عنوان اولین پارامتر **mySqlCommand**درج کرده ایم.

مجموعه**System.Data.SqlDBType** نیز حاوی انواع داده ای موجود در **SQL-Server** است.

برای مقدار دهی این پارامتر هانیز کافی است از دستوراتی به شکل زیر استفاده کنید:

mySqlCommand.Parameters[“@myTrmno”].Value=’3831’;

مثال1:فرض کنید می خواهیم متدی بنویسیم که یک شماره دانشجویی ویک شماره ترم ر ابه عنوان پارامتر بگیرد و تمام رکوردهای موجود برای این شمار ه دانشجویی در ترم مذکور را در فایل ثبت نام را حذف کند.

در این مثال درخواستی که برای پایگاه داده فرستاده می شود نتیجه خروجی ندارد بنابر این می توانیم از**DataReader** استفاده نکنیم وتنها از **SqlCommand** کمک بگیریم:

```
private void reg_del(int snum,string termnum)
{
        string mySqlConnectionStr = "workstation
id=[Local];packet size=4096;integrated
security=SSPI;initial catalog=registration;persist security
info=False";
       string mySqlQuery = "delete * from Reg where s#=@snum
and trmno=@termnum";
        SqlConnection mySqlConnection = new SqlConnection();
        mySqlConnection.ConnectionString=mySqlConnectionStr;
        mySqlConnection.Open();
        SqlCommand mySqlCommand = new SqlCommand();
        mySqlCommand.Connection=mySqlConnection;
        mySqlCommand.CommandText=mySqlQuery;
        mySqlCommand.Parameters.Add("@snum",
System.Data.SqlDbType.Int,7);
        my
SqlCommand.Parameters.Add(“@termnum”,System.Data.SqlDbTy
pe.char,4);
        mySqlCommand1.Parameters["@snum"].Value=snum;
        // or you can code
mySqlCommand1.Parameters[0].value=snum
        my Sql
Command.Parameters.[“@termnum”].Value=termnum;
        mySqlCommand1.ExecuteNonQuery();
        mySqlConnection1.Close();

}
```

مثال2:برای آشنایی با متد **ExecuteScalar**در این مثال تابعی می نویسم که تعداد دانشجویانی که معدل کل آن ها دریک ترم مشخص که شماره آن ترم(یک مقدار رشته ای) به عنوان پارامتر برای این تابع پاس می شود بالاتر از 17 می باشدرا بر گرداند.

```
private void int reg_del(string termnum)
{
```

```
        string mySqlConnectionStr = "workstation
id=[Local];packet size=4096;integrated
        security=SSPI;initial catalog=registration;persist security
info=False";
        int myResult;
        string mySqlQuery = "select count(*) from (select distinct s#
form stdtrm”+
        “where trmgpa>17 and trmno=@termnum”
        SqlConnection mySqlConnection = new SqlConnection();
        mySqlConnection.ConnectionString=mySqlConnectionStr;
        mySqlConnection.Open();
        SqlCommand mySqlCommand = new SqlCommand();
        mySqlCommand.Connection=mySqlConnection;
        mySqlCommand.CommandText=mySqlQuery;
        my
SqlCommand.Parameters.Add(“@termnum”,System.Data.SqlDbTy
pe.char,4);
        my Sql
Command.Parameters.[“@termnum”].Value=termnum;
        myResult = mySqlCommand1.ExecuteScalar();
        mySqlConnection.Close();
        return myResult;
}
```

مثال3:برای بررسی تقاضای اجرای دستوری که نتیجه خروجی آن بیش از یک رکورد باشد متدی برای کلاس فرم **std**

می نویسیم که یک شماره دانشجویی را به عنوان پارامتر در یافت کند وشی**myStd**که شیئی از کلاس فرم **std**است را با اطلاعات دانشجوی مذکور پر کند.(البته پیاده سازی این تابع به روش ساده تری هم امکان پذیر است که با توجه به هدف آموزشی این مثال به روش زیر آن را پیاده سازی می کنیم)

```
private void int reg_del(int snum)
{
    string mySqlConnectionStr =...
    string mySqlQuery1 = "select * from std”;
    / S#, Name,Family,Field,Sex
,Totpassunit,Totregunit,Gpa,Address,Citycode, Telno,
    Ssno,Birthdate
```

```
    SqlConnection mySqlConnection1 = new SqlConnection();

mySqlConnection1.ConnectionString=mySqlConnectionStr;
        mySqlConnection1.Open();
        SqlCommand mySqlCommand1 = new SqlCommand();
        mySqlCommand1.Connection=mySqlConnection1;
        mySqlCommand1.CommandText=mySqlQuery1;
        SqlDataReader mySqlDataReader1 =
mySqlCommand1.ExecuteReader();
        while ( mySqlDataReader1.Read())
        {
          If (mySqlDataReader1.GetInt32(0)=snum)
          {
                myStd.Snum.text=mySqlDataReader1.GetString(0);
                myStd.Name.text=mySqlDataReader1.GetString(1);
                myStd.Family.text=mySqlDataReader1.GetString(2);
                ……
            }
        }
    mySqlDataReader1.Close();
       mySqlConnection1.Close();
}
```

مثال4:در این مثال قطعه کدی نوشته می شود که نحوه تقاضای اجرای یک رویه ذخیره شده رانشان می دهد.این رویه همان اولین رویه ای است که در آزمایش سوم نوشتیدیعنی رویه **TrmGpa_PR**که شماره یک دانشجو و شماره ترم رادریافت نموده و معدل ترم دانشجو را محاسبه و برمی گرداند.این رویه را با پارامتر های شماره دانشجویی 8017062 و ترم 3821 احضار می کنیم وخاصیت **text**مر بوط به شی **Label1**ر ابا خروجی این رویه مقدار دهی می کنیم.

```
        string mySqlConnectionStr =…
        SqlConnection mySqlConnection1 = new
SqlConnection();

mySqlConnection1.ConnectionString=mySqlConnectionStr;
        mySqlConnection1.Open();
        SqlCommand mySqlCommand1 = new SqlCommand();
        mySqlCommand1.Connection=mySqlConnection1;
```

```
mySqlCommand1.CommandType=System.Data.CommandType.StoredProcedure;
        mySqlCommand1.CommandText=”CR_TrmGpa”;
        mySqlCommand1.Parameters.Add("@snum", System.Data.SqlDbType.Int,7);
        my SqlCommand.Parameters.Add(“@termnum”,System.Data.SqlDbType.char,4);
        mySqlCommand1.Parameters["@snum"].Value=8017062;
        mySql Command.Parameters.[“@termnum”].Value=”3821”;
        SqlDataReader mySqlDataReader1= cmd.ExecuteReader()
       If (mySqlDataReader1.Read())
       label1.text=mySqlDataReader.Getstring(0);
       else
       Label1.Text=("No Record found")
       mySqlDataReader.Close();
       mySqlConnection1.Close();
```

معرفی شی های **DataSet**و**DataAdapter**:

دیدیم که می توان با استفاده ازمتد های اشیا**Connection**و**Command**و**Datareader**درخواست ها را به موتور پایگاه داده ها فرستاد و نتایج این در خواست ها فقط دریافت نمود .حال اگر بخواهیم این نتایج را در جایی ذخیره کنیم نیاز به شی جدیدی خواهیم داشت که در محیط **Net.**این شی **DataSet**می باشداین ساختار یک ساختار تشکیل شده در حافظه است.**DataSet**در واقع نگهدارند نتایج در خواست هاست ولی نمی داند نتایجی که درون خود دارد از کجا آمده است زیرا این شی از منبع داده ای خود جداست .**DataSet**را می توان به جای دسترسی مستقیم به خود **DataBase**مورد استفاده قرار دادو در صورت تغییر **DataSet**می توان تغییرات را به پایگاه داده ها منتقل کرد. این شی را می توان یک پایگاه داده ای رابطه ا ی که از تعدادی جدول و روابط بین این جداول تشکیل شده است در نظرگرفت.برای ایجاد داده در **DataSet**دو راه عمده وجود دارد

1. در درون **DataSet**جداول جدیدی ایجاد کنیم.

2. **DataSet**را با نتایج حاصله از یک **Select**که خود در واقع تعدادی جدول است پر کنیم.

نقش **DataAdapter**در این جا این است که به کمک آن داده های در خواست ها را از پایگاه داده گرفته و **DataSet**را از نتایج این در خواست ها پر (**Fill**)می کند . **DataAdapter**(با کمک **DataConnection**)در واقع یک مدیرداده ای است که هم جریان داده از **DataSet**به سمت منبع داده و هم جریانی با جهت برعکس را کنترل می کند.**DataAdapter**در واقع در کنار **DataSet**معنای واقعی خود را پیدا می کند.**DataAdapter**برای اعمال ذخیره وبازیابی اطلاعات می تواند از شی**Command** استفاده کند.

DataSetاز دو شی اصلی تشکیل شده است:

1-**DataTable**:این شی امکان دسترسی به تاپل ها و ستون های یک جدول رافراهم می کند.

2- **DataRelation**:با استفاده از این شی می توان ارتباط بین جداول را تعریف کرد.

مثال: مثال زیربرای آشنایی بیشتر با اشیا **DataSet** و**DataAdapter**وکنترل**DataGrid**مناسب خواهد بود.

1. فرم جدیدی را به پروژه اضافه کنید.خاصیت نا م را به**std2** وخاصیت **text**آن را به"فرم آدرس دانشجویان" وخاصیت**RightToLeft**آن را به **Set،yes**کنید.
2. یک کنترل**DataGrid**به فرم اضافه به نحوی که این **DataGrid** تمامی سطح فرم را بپوشاند.
3. حال از **ToolBox**گزینه (تب) دیتا را انتخاب و یک **SqlDataAdapter**به فرم اضافه کنید.
4. مراحل ایجاد ویزارد **SqlDataAdapter**را طی کنید.**Connection**را به **Registration**بزنیدو **QueryType** را **UseSqlStatements**انتخاب کنید.می خواهیم**DataGrid**مور نظر شامل ستون های نام ،نام خانوادگی وآدرس دانشجویان بوده و بر اساس نام خانوادگی و نام مرتب شده باشد.بنابر این **SqlStatement** با عبارت زیر مقدار دهی کنید:

```
Select name,family,Address
From std
Order by family,name
```

بعد از پایان ویزارد میبینید که علاوه بر **DataAdapter**یک **connection**هم به فرم اضافه می شود . چون **DataAdapter** از یک **Connection**برای ارتباط با پایگاه داده **registration**استفاده می کند.

5. حال باید جایی برای ذخیره نتایج تعیین کرد.بنابراین از گزینه**Data**،گزینه **generateDataSet**راانتخاب کنبد.نام این **DataSet**را **myds**انتخاب و خصوصیات آن را مقدار دهی کنید.
6. حال ما یک مجموعه نتیجه حاصل از یک درخواست و جایی برای ذخیره سازی داریم .تنها کاری که باید کرد این است که عملا اعلام کنیم این نتایج باید در این محل ذخیره شود.برای این کار باید از **DataAdapter**خواست تا **DataSet**ر ابا مقادیر خروجی مورد نظر پرکند.باید بااستفاده از متد **fill**یک **DataAdapter**این کار ر اانجام داد.بنابر این کد زیر را برای انجام این کار در هنگام مثلا لود شدن فرم بنویسید.

sqlAdapter1.Fill(myds);

7. خوب تا این جا داده های ما در **DataSet**ذخیره شده اند ومی خواهیم آن ها را نمایش دهیم.کنترلی که می تواندآئینه نمایش دهنده محتویات یک جدول باشد **DataGrid**است .حال باید تعیین کرد **DataGrid**داده های خود را از کجا بیاورد بنابر این خاصیت **DataSource** و**DataMember**(**DataSouce**نمایانگر**DataSet**مرجع و **DataMember**نمایانگر جدولی از **DataSet**می باشدکه می خواهیم عناصر آن نمایش داده شوند.) آن ها را مقدار دهی کرده و برنامه را اجرا کرده و نتیجه را مشاهده کنید.

تعریف **DataAdapter**ها:

برای تعریف **DataAdapter**ها می توا ن از روش های متفاوتی در مد کدنویسی استفاده کرد .

1. مثال:

```
String strConn =
"Server=(Local);Database=registration;integrated security=true;"
    SqlConnection cn = new SqlConnection();
    Cn.ConnectionString = strConn;
    SqlDataAdapter myStdDA,myRegDA;
    myStdDA = New SqlDataAdapter("SELECT * FROM Std", cn)
    myRegDA = New SqlDataAdapter("SELECT * FROM Reg", cn)
```

می بینید که در یکی از شکل های استفاده از سازنده یک **SqlDataAdapter** می توان دستور **select** آن **DataAdapter** را به عنوان پارامتر اول و**Connection**ای که آن **DataAdapter**از آن استفاده می کند ر ابه عنوا ن پارامتر دوم به سازنده پاس کرد .

دقت کنید که این دستور **Select**نماینده مجموعه نتایجی است که یک**DataAdapter**بر می گرداند(به شکل یک جدول)که می توان آن را در یک **DataSet**ذخیره کرد.

2. می توان ازیک **DataComand**هم کمک گرفت :

مثال:

```
String strConn =
"Server=(Local);Database=registration;integrated security=true;"
SqlConnection cn = new SqlConnection();
Cn.ConnectionString = strConn;
String strSQL  = "SELECT Name,Family FROM Std";
SqlCommand cmd = new SqlCommand(strSQL, cn);
SqlDataAdapter myStdDA=new
SqlDataAdapter(cmd) ;
```

در این حالت **DataAdapter**حاوی مقادیر حاصل از اجرای دستورات موجود در **CommandText**است.

پر کردن یک **DataSet** بوسیله **DataAdapter**:

مثال:قطعه کد قبلی را در نظر بگیریدحال یک **DataSet**تعریف می کنیم و آن را پر می کنیم:

```
DataSet()  ds = new DataSet();
myStdDA.Fill(ds);
```

سپس مثلا برای استفاده از این **DataSet**به نحو زیر عمل میکنیم .فرض کنید یک **DataGrid**با نام **DataGrid1**در سطح فرم داریم:

```
DataGrid1.DataSource = ds;
DataGrid1.DataMember = "Table";
```

گفتیم که خصوصیت **DataMember**از یک **DataGrid**باید با نام جدولی از **DataSet**که می خواهیم توسط**DataSet** نمایش داده شود **Set**شود**.**

نکته مهمی که برای مشخص کردن جداول موجود دریک **DataSet** به عنوان منابع داده ای وجود دارد این است که اگر برای جداولی که در نتیجه اجرای دستورات **DataAdapter**وارد یک **DataSet**می شوند (با اجرای متد **Fill**از**DataAdapter**)نامی مشخص نکنیم اولین جدول به نام **Table**دومی**Table1**والی آخر نام گذاری می شوند.برای تغییر نام نتایج حاصل از اجرای **DataAdapter**به نام دلخواه از یکی ازمجموعه های هر **DataSet**به نام **TableMapping** استفاده می شود.وظیفه این مجموعه تغییرنام جداول موجود در **DataSet**به نام های دلخواه کاربر جهت سهولت کار کردن با آن ها است.مثلا برای تغییر نام جدول موجود در **DataSet**با نام **Table**از دستور زیر استفاده می کنیم:

```
da.TableMappings.Add("Table", "Std")
```

در اینجا خواص **DataGrid1**نیز به شکل زیر مقداردهی میشد:

```
DataGrid1.DataSource = ds;
DataGrid1.DataMember = "Std";
```

روش بهتری برای تعیین نام جدول مجموعه نتایج در یک **DataSet** در هنگا م پر کردن **DataSet**به شکل زیراست:

```
myStdDA.Fill(ds, "Std")
```

در این صورت مجموعه نتایج حاصل از اجرای دستورات **DataAdapter**در جدولی با نامی که درپارامتر دوم متد **Fill**یک**DataAdapter**مشخص شده است قرار می گیرد.

نکته :

- **DataSet**اگر چه در هر **DataSet** نام جدول منبع بانام جدول درونی به طور پیش فرض یکی نیست ولی نام ستون های در خواست شده در **query**وجدول موجود در **DataSet**یکسان است.مثلا جدول **Std**موجود در**ds**ازدو ستون به نام های **Name**و **Family** تشکیل شده است.
- چون در این جا در مورد خواص **Datasource**و**Datamember**صحبت کردیم این نکته را هم را ذکر

می کنیم که در مورد کنترل هایی که حالت لیستی به خود می گیرند(مثلا **ListBox**یا**ComboBox** نیز این امکان که یک **DataSource** و**DataMembr**ر ا**Set**کنیم وجود دارد .مثلا این که عناصر یک **listBox** شامل بر نام درس های یک دانشکده خاص باشد.برای این کار باید نام دروس آن دانشکده را در جدولی درون **DataSet**ریخته و این جدول رل به عنوان منبع داده ای آن **listBox**،تنظیم کنیم.

بررسی دقیق تر **DataSet**:هر**DataSet**دارای دو**Collection**به نام های **DataRow**و**DataColumn** است.بعد از اجرای درخواست هر **DataAdapter**،برای هر ستون مجموعه نتایج یک شی **DataColumn**ساخته می شود.دو خاصیت مهم هر **DataColumn**،**Name**و**DataType**است که برای هر شی **Datacolumn**توسط **DataAdapter**مقدار دهی می شود.

مثال :در مثال زیر با نام و نوع هر ستون از اولین جدول **ds**کار می کنیم:

```
DataTable tbl = ds.Tables(0);
Label1.text=tbl.tableName;
DataColumn col;
```

```
textBox1.text=tbl.Columns(0).ColumnName;
textBox2.text=tbl.Columns(0).DataType.Tostring;
textBox3.text=tbl.Columns(1).ColumnName;
textBox4.text=tbl.Columns(1).DataType.Tostring;
```

DataTableیک نوع داده ای(**DataType**) است که به متغیر های از نوع جدول اشاره می کند.
بعد از اجرای کد بالا متغیر ها ی استفاد شده مقادیر زیر را خواهند داشت:

```
Label1.text=Std;
textBox1.text=Name;
textBox2.text=varchar(16);
textBox3.text=Family;
textBox4.text=varchar(30);
```

اما مهم ترین مجمو عه موجود در **DataTable**،مجموعه**DataRow**است که با استفاده از آن می توان به هر سطر از جدول دسترسی داشت.برای استفاده از این مجموعه باید از خصوصیت **Rows**از **DataTable**کمک گرفت.
مثال:مجموعه دستورات زیر **textBox1.text**ر ابا نا م و **textBox2.text**را با نام خانوادگی دانشجوی پانزدهم از جدول **Std**مقدار دهی می کند:

```
DataTable myTable = ds.Tables(0);
DataRow myDataRow=myTable.Rows(14)
textBox1.text =myDataRow(“Name”);
textBox2.text = myDataRow(“Family);
```

دقت کنید اندیس **Rows**از 0 شروع می شود.

نحوه اضافه کردن یک جدول خام به یک **DataSet**:
برای اضافه کردن یک جدول(متغیری از جنس **DataTable**)به یک **DataSet**از متد **Add**از **DataSet**استفاده می کنیم :
مثال:

```
DataTable myTable  = ds.Tables.Add("teachers");
```

در این مثال جدول **myTable**به **ds**اضافه شده است .نام جدول **myTable**در **teachers.ds**است.
متد **Add**با کاربرد مشابه در مجموعه های دیگری نیز موجود است مثلا در **columns**می توان از آن برای اضافه کردن یک ستون به یک جدول استفاده کرد.

```
DataColumn = tbl.Columns.Add("teacherId",GetType(int));
```

تا این جا **DataSet**ر ادر حد نیاز در این آزمایشگاه بررسی کردیم. تنها مطلبی که در این جا ذکر می کنیم این است که یک**Dataset**از منبع داده ای خود خبر ندارد وفقط یک کپی از داده هارا در خوددارد و نمی توان مثلا انتظار داشت که اگر چند جدول که در پایگاه داده دارای روابط ارجاعی بودند و با استفاده از **DataAdapter**های مختلف به درون یک **DataSet**کپی شدند آن **DataSet**نیز از این روابط ارجاعی آگاه باشد و در صورت درج ،حذف یا تغییر داده این روابط در نظر گرفته شود.**DataSet**حتی از محدودیت های موجود بر روی یک جدول (به جز مثلا انواع داده ای ستون ها)نیز آگاه نیست.برقراری روابط ارجاعی بین جداول یک**DataSet**بوسیله شی **DataRelation**صورت می گیرد کهدر این آزمایشگاه بررسی نخواهد شد.

به روزرسانی یک منبع داده ای :

برای این که تغییراتی که در داده های **DataSet**داده می شوند در منبع داده ای اصلی نیز به روز رسانی شوند باید از متد **Update**مر بوط به **DateAdapter**ی که آن داده راا از منبع داده ای اولیه به **DataSet**منتقل کرده استفاده کنیم.این متد را می توانید به شکل زیر به کار ببرید :

Update(DataSet,TableName)

بعد از صدا زدن این متد، **DataAdapter**سطر به سطر داده های جدول ذکر شده در **DataSet**را بررسی می کند و در صورتی که تغییراتی روی سطر مورد نظر داده شده باشد آن را به منبع داده ای منتقل می کند.**DataAdapter**این کار را این گونه انجام می دهد که ابتدا نوع تغییری که روی سطر مربوطه داده شده است(درج ،اصلاح یاحذف) را تشخیص می دهد و سپس یکی از متد های **InsertCommand**یا**DelateCommand**یا**UpdateCommand** خود رااحضار می کند.اما **CommandText**این **Command**ها چگونه است ؟جواب به این سوال این است که بعد از این که متغیر**Command**یک **DataAdapter**مقداردهی شد بلافاصله این توابع نیز بدنه خود راپیدا می کنند .بدین صورت که اگر یک **DataAdapter** به عنوان مثال نتایج پرس و جوی زیر را از منبع داده ای بگیرد:

```
Select S#,name,Family
From  Std
```

آن گاه به عنوان مثال **UpdateCommand**آن به شکل زیر است:

```
UPDATE STD SET
S# = @S#, Name = @Name, Family = @Family
WHERE (S# = @Original_S_)
AND (Family = @Original_Family)
AND (Name = @Original_Name);
```

برای هر سطر **Update**شده درون جدول یک**DataSet**متغیر **#S@**مقدار جدید **#S**و **@Original_S_**مقدار **#S**قبل از تغییر را نگهداری می کند.می توانیداین **Command**ها را خودتان مقدار دهی کنید.از این روش برای جلوگیری از بازتاب تغییر مثلا تغییر بعضی از ستون هادر منبع داده ای استفاده می شود.

مثال:

```
myDataAdapter.UpdateCommand=
"UPDATE STD SET
Name = @Name, Family = @Family
WHERE (S# = @Original_S_)
```

بررسی متد **Select**از یک **DataTable**:

از این متد برای فیلتر کردن سطر های یک **DataTable**در زمان اجرا بدون تحت تاثیر قرار گرفتن خود جدول استفاده می شود.

مثال :فرض کنید می خواهیم روی یک **ListBox**در یک فرم تمامی شهر های موجود در پایگاه داده **registration**نمایش داده شود تا کاربر بتواند از میان آن ها یکی را انتخاب کند.قطعه کد زیر این کار را انجام می دهد(فرض می کنیم محتویات کل جدول **CodeFile**قبلابوسیله یک **DataAdapter**درجدولی به نا م**Codefile**و در دیتاستی با نام **myDataSet**ریخته شده است همچنین فرض می کنیم **myListBox** یک **ListBox**از قبل تعریف شده است. **)**

```
DataRow[] mySelectedRows;
mySelectedRows=myDataSet . Codefile .select("field= cityCode");
foreach(DataRow myDataRow in mySelectedRows)
{
  myListBox.Items.Add(myDataRow["[Desc]"]
}
myListBox.Refresh();
```

دستور**foreach**برای دسترسی به عناصر یک مجموعه به کار می رود.

بررسی **DataBinding**وکاربرد آن در فرم های ویندوزی:

در این جا می خواهیم این موضوع را بررسی کنیم که چگونه می توان یک فرم را به اطلاعات یک جدول که در **DataSet**وجود دارد **Bind**کرد به گونه ای که با حرکت روی تاپل های مختلف آن جدول اطلاعات آن تاپل در کنترل های مختلفی که روی فرم قرار دارد نمایش داده شوند.برای این کار از دستوری مثل دستور زیر استفاده میکنیم:

```
snum.DataBindings.Add("Text",myDataSet,"std.s#");
```

فرض کنید **myDataSet**حاوی جدولی با نام**Std**است که با استفاده از یک **DataAdapter**ازاطلاعات جدول **Std**موجود در پایگاه داده **registration**پر شده است. **Snum**نیزمقدار خصوصیت نام یک **textBox**بود که میخواهیم شماره دانشجویی درون آن قرار گیرد.با استفاده از دستور بالا شماره دانشجویی اولین دانشجو از جدول **Std**درون جعبه متنی **Snum** قرار می گیرد.

حالا که می توانیم کنترل های فرم های ویندوزی را به ستون ها ی جداول موجوددر **DataSet**،**bind**کنیم کافی است روی تاپل های مختلف جدول حرکت کرده و داده های آ ن تاپل را روی کنترل های فرم به نمایش در آوریم.

برای این کار **Net Framework.** شی **BindingContext**رادر نظر گرفته است .با استفاده از این شی می توانیم علاوه بر حرکت روی سطر های مختلف یک جدول،کنترل های فرمی رابا محتویات ستون های آن سطر داده ای پر کنیم .

متد مهمی برای این شی وجود ندارد ولی دو خصوصیت مهم این شی عبارند از :

1. **Position**:در حین حرکت روی تاپل های یک جدول شماره سطر را نگه می دارد.
2. **Count**:تعدادتاپل های جدول را نمایش می دهد.

فرض کنید تعدادی از تاپل های یک جدول را خوانده ایم و می خواهیم ببینیم هم اکنون روی کدام تاپل هستیم:

می توان ازدستور زیردرداخل فرمی که به مثلا به جدول **Std**درون **myDataSet**،**DataSet** **bind**شده استفاده کرد:

```
int myPosition = std_Form .BindingContext[myDataSet,"Std"]
.Position;
```

می بینید که شی **BindingContext**برای فرم تعریف می شود.

مثال:در این مثال فرم**Std**را کامل می کنیم و بعضی از رویداد های آن را کد نویسی می کنیم.بررسی این مثال روند استفاده از شی **BindingContext**را روشن می کند:

ابتدا متغیر های سطح فرم زیر را برای استفاده ازآن ها در کلیه متد های فرم تعریف می کنیم:

```
string   myStrSql;
string   myStrCon;
int      myIndex;
SqlDataAdapter   myDataAdapter;
SqlConnection   myCon;
```

کدهای زیر را به ابتدای متد کنترل کننده رویداد لود شدن فرم**Std**اضافه کنید:

```
private void Form1_Load(object sender, System.EventArgs e)
{
```

```
  myStrCon="workstation id=(DBLAB-2);packet
size=4096;integratedsecurity=SSPI;initial
catalog=registration;persist security info=False";
  myStrSql="select * from std";
  myCon = new SqlConnection();
  myCon.ConnectionString=myStrCon;
  myCon.Open();
  myDataAdapter = new SqlDataAdapter(myStrSql,myCon);
   myDataAdapter.Fill(myDataSet,"Std");
```

تا این جا جدول **Std**را(برای **bind**شدن با فرم **Std**در **Load**،**DataSet**کرده ایم:
حال بایدکنترل های مربوطه را با ستون های جدول **Std**،**Bind**کنیم**.** بنابر این قطعه کدهای زیر را به انتهای **Form1_Load**اضافه می کنیم:

```
  snum.DataBindings.Add("Text",myDataSet,"std.s#");
  name.DataBindings.Add("Text",myDataSet,"std.name");
  family.DataBindings.Add("Text",myDataSet,"std.family");
  ssno.DataBindings.Add("Text",myDataSet,"std.ssno");
  address.DataBindings.Add("Text",myDataSet,"std.address");
  city.DataBindings.Add("Text",myDataSet,"std.citycode");
  telno.DataBindings.Add("Text",myDataSet,"std.telno");
  bdate.DataBindings.Add("Text",myDataSet,"std.birthdate");
  gpa.DataBindings.Add("Text",myDataSet,"std.gpa");

totpassunit.DataBindings.Add("Text",myDataSet,"std.totpassunit");
  totregunit.DataBindings.Add("Text",myDataSet,"std.totregunit");
```

این خطوط کد کلیه فیلد هایی را که عیناو بدون هیچ تغییری باید نمایش داده شوندرا **bind**می کند ولی ضعف عمده روش **Databinding**این است که فیلد هایی که باید با تغییراتی روی فر م به نمایش در آیند را نمی تواند **bind**کند.برای حل این مشکل دو راه حل وجود دارد.روش بهتر این است که مثلا در این فرم چون نمیتوان مستقیما نام شهر دانشجو را از جدول **Std**استخراج کرد واستفاده از جدولی غیر از **Std**لازم است واین فیلد باید ازفایل**codeFile** استخراج شود. راه حل این است که به کمک یک **DataAdapter**جدولی در **myDataSet**ایجادکنیم که حاوی کد شهر ها و شهر های مربوطه بوده و سپس یک **Relation**بین این 2 جدول(**STD**وجدول حاوی کدونام شهرها) تعریف کنیم وبوسیله تعریف یک **DataBinding**روی این **Relation**داده ها رابه فرمت دلخواه کاربر در آوریم.در این روش بعد از گرفتن داده ها از کاربرونیز قبل از وارد شدن در پایگاه داده،داده ها به فرمت

مورد قبول پایگاه داده در می آیند.ولی چون از معرفی شی**DataRelation** چشم پوشیدیم از روش دومی استفاده می کنیم .در این روش با استفاده از **Position**واستفاده از یک **DataCommand**به فیلد موردنظر با فرمت دلخواه دسترسی پیدا کرده وآن را روی فرم به نمایش در می آوریم .قطعه ای از کد زیر این موضوع را روشن می کنداین کد را به انتهای **Std_Load**اضافه کنید.:(**index**یک **TextBox**است که در هر لحظه شماره تاپلی که روی آن قرار داریم از جدول **Std**راروی فرم به نمایش در می آورد.)

```
1. myIndex=this.BindingContext[myDataSet,"Std"].Position;
2. index.Text=Convert.ToString(myIndex);
3. if
   (System.Convert.ToChar(this.myDataSet.Tables["Std"].Rows[
   myIndex]["Sex"])=='M')
4. radioButton1.Checked=true;
5. else
6. radioButton2.Checked=true;


7. SqlCommand mySqlCommand1 =new SqlCommand();
8. mySqlCommand1.Connection=myCon;
9. mySqlCommand1.CommandText="select codefile.[desc] from
   std,codefile where "+
10."codefile.field = 'Citycode' "+ "and Std.S#= " +
11.System.Convert.ToString(this.myDataSet.Tables["Std"].Rows
   [myIndex]["S#"])+
12." and codefile.Type='"+
        System.Convert.ToString(this.myDataSet.Tables["Std"].
   Rows[myIndex]
13.["Citycode"])+" ' ";
14.SqlDataReader myDataReader1=
   mySqlCommand1.ExecuteReader();
15.myDataReader1.Read();
16.city.Text=System.Convert.ToString(myDataReader1["desc"]);

17.myDataReader1.Close();
18.mySqlCommand1.CommandText="select codefile.[desc] from
   std,codefile      where "+"codefile.field = 'field' "+ "and
   Std.S#= " +
        System.Convert.ToString(this.myDataSet.Tables["Std"].
```

```
Rows          [myIndex]["S#"])+" and codefile.Type='"+

        System.Convert.ToString(this.myDataSet.Tables["Std"].
Rows      SqlDataReader myDataReader1=
mySqlCommand1.ExecuteReader();
19.myDataReader1.Read();
20.field.Text=System.Convert.ToString(myDataReader1["desc"])
;
21.myDataReader1.Close();
22.myCon.Close();
```

توضیحات لازم در مورد قطعه کد بالا:

از متغیر**myIndex**برای نگهداری شماره سطر درحین حرکت بین تاپل ها استفاده می کنیم .در خطوط 2تا6بااستفاده از متغیر **myIndex**جنسیت دانشجوی فعلی را بدست می آوریم و **radioButton**مربوطه را **Check**می کنیم.درهمین جایک روش دسترسی به مقدار یک ستون خاص را می بینید:

this.myDataSet.Tables["Std"].Rows[myIndex]["Sex"]

در خطوط 7 تا 17 با استفاده از اشیاء **DataReader**و **DataCommand** و استفاده از **myIndex**به رشته دانشجویی که روی سطر مربوط به وی قرار داریم دسترسی پیدا می کنیم.قبلا گفتیم که ارسال پارامتر ها به **Command** دوروش دارد یک روش را قبلا مطرح کردیم روش دوم را در دستور 18 دیده می شود که در این روش پارامتر هارا به متغیرهایی از نوع **string**، **Cast**می کنیم و سپس آن ها را به خاصیت **text**از **Command**مورد نظراضافه می کنیم .سپس با اجرای **Command**نتایج آن رادرون یک **dataReader**می ریزیم.در دستور 16 نحوه دسترسی به یک ستون از تاپلی که یک **dataReader**روی آن قرار دارددیده می شود.

myDataReader1["desc"]);

در دستور17 **dataReader**ر ابسته ایم زیرا در دستورات بعدی می خواهیم همین **dataReader**ر ابا **Command**متفاوتی مورد استفاده قرار دهیم .

در دستورات 18 تا 22 نیز از روش مشابهی برای استخراج شهر دانشجو استفاده کرده ایم.در پایان نیز **Connection**را بسته ایم.

می توانیم دستورات 7تا 23 را در قالب یک متد پیاده سازی کنیم .زیرا این دستورات در جاهای دیگری نیز مثلا در متد های **Handle**کننده کلید هایی که باعث می شوند یک رکورد رو به جلو یا رو به عقب حرکت کنیم یا کلید هایی که ما را روی رکورد اول یا آخر می برند،نیز عینا استفاده می شوند.زیرا در هنگام روی دادن این رویداد ها نیز باید تبدیل داده ای صورت بگیرد.

در پایان برای آشنایی بیشتردانشجویان ، متد اداره کننده کلیک بر روی کلیدی که شمارنده را روی رکورد ها یکی یکی رو به جلو می برد ،کد می کنیم:

```
private void nextRecord_Click(object sender, System.EventArgs e)
{
myCon.Open();
myIndex = this.BindingContext[myDataSet,"Std"].Position+1;
if ( myIndex<this.BindingContext[myDataSet,"Std"].Count)
{
  this.BindingContext[myDataSet,"Std"].Position +=1;

index.Text=Convert.ToString(this.BindingContext[myDataSet,"Std
"]
  .Position+1);
  index.Text=Convert.ToString(myIndex);
  if
(System.Convert.ToChar(this.myDataSet.Tables["Std"].Rows[myIn
dex]
  ["Sex"])=='M')
              radioButton1.Checked=true;
  else
              radioButton2.Checked=true;
  SqlCommand mySqlCommand1 =new SqlCommand();
  mySqlCommand1.Connection=myCon;
  mySqlCommand1.CommandText="select codefile.[desc] from
std,codefile where       "+"codefile.field = 'Citycode' "+ "and
Std.S#= " +

System.Convert.ToString(this.myDataSet.Tables["Std"].Rows[myIn
dex]
  ["S#"])+
  " and codefile.Type='"+
   System.Convert.ToString(this.myDataSet.Tables["Std"].
   Rows[myIndex]["Citycode"])+"'";
   SqlDataReader myDataReader1=
mySqlCommand1.ExecuteReader();
   myDataReader1.Read();
   city.Text=System.Convert.ToString(myDataReader1["desc"]);
```

```
   myDataReader1.Close();
        .
      .
         .
myCon.Close();
}
}
```

باتوجه به مطالب گفته شدهروشن شد که **ADO.Net**برای عملیات گوناگون ابزارهای متفاوتی را در اختیاربرنامه نویس قرار می دهد .در این جا می خواهیم دوروش متفاوت پیاده سازی یکی از این عملیات ها یعنی به روز رسانی یک سطر را مرور وموارد کاربرد آن ها را مقایسه کنیم:

1-یک روش برای پیاده سازی تغییرات نوشتن یک **Command**مناسب (بسته به نوع عمل (**insert**یا**delete**،**update**))

است.در این جا می توانیم مقادیر موجود در روی کنترل ها را بخوانیم ودر منبع داده ای سطری که متغیر **position**به آن اشاره می کند را به مقادیر جدید کنترل ها **Set**کنیم.در این روش چون **DataSet**،به روز رسانی نمی شود کافی است با استفاده از متد **()clear** یک **DateSet**آن را خالی کنیم و دوباره آن را پر کنیم.پیشنهاد می کنیم متد هایی برای پر کردن هر **DataSet**با توجه به جدول هایی که دارد بنویسید.

2-قبل از توضیح روش دوم این نکته را ذکر می کنیم که هرشی **DataTable**متدی به نام **()acceptchanges** دارد. تغییراتی که روی یک جدول داده می شوند در چیزی مانند تاریخچه نگه داشته می شود .تنها هنگامی که متد **()acceptchanges**صدا زده می شود تغییرات داده شده به صورت واقعی بر روی جدول اعمال می شود والبته دیگر این تغییرات قابل برگشت نیست.هر گاه متد **()update**از یک **DataAdapter** رابرای به روز رسانی منبع داده ای یک جدول در **DataSet** احضار شود این متد ،متد **()acceptchanges**جدول مربوط به خود را نیزاحضار می کند.

زمانی که از **DataBinding**برای اتصال فرم به یک **DataSet**استفاده می کنیم .بعد از تغییرروی فرم که به **DataSet**هم اعمال می شودکافی است متد **Update**از**dataAdpater**مربوطه را صدا بزنید.در این حالت کاربرمی تواند از اعمال این تغییرات روی منبع داده قبل ازاحضار متد **()Update**منصرف شود.در این صورت تغییرات در حد همین **session**اتصال به **DataSet**باقی خواهد ماند.

اما یک نکته دیگر نیز در این حالت مطرح است وآن این است که فیلد هایی را که **bind**نیستند(مثل فیلد کد شهر در فرم**Std**)را در صورت تغییر، چگونه به روز رسانی کنیم .برای این کار نیز دو راه حل قابل طرح است.

1-استفاده از **DataCommand**.

2-اعمال تغییرات هم زمان با تغییر به روی **DataSet**:در این حالت در صورتی که کاربر تغییری روی فرم بدهد،می توان با استفاده از متغیر **position**آن سطر داده را در **DataSet**به روز رسانی کرد.مثلا اگر کاربر جنسیت دانشجویی که متغیر **Position**به او اشاره می کند را تغییر داد !می توان با استفاده از دستور زیر این تغییر را در **DataSet**اعمال کرد:

```
myIndex = this.BindingContext[myDataSet,"Std"].Position;
myDataSet.Tables[“Std”].Rows[myIndex][Sex]=’M’;
```

یا به صورت کامل تر می توان نوشت:

```
If(readiButton1.checked())
  myDataSet.Tables[“Std”].Rows[myIndex][Sex]=’M’;
else
  myDataSet.Tables[“Std”].Rows[myIndex][Sex]=’F’;
```

میبینید که می توان بدون بررسی این که کاربر تغییراتی ر ااعمال کرده است یا نه **DataSet**ر ابه روز رسانی کرد.

می توانستید این کار رابه چند صورت ساده تر انجام دهید. مثلا تنها در رویداد **radioButton1_CheckedChanged**این به روز رسانیِ **DataSet**را انجام دهید.

بین دوروش اخیر مسلما روش اعمال تغییرات هم زمان با تغییر به روی **DataSet** از روش استفاده از **DataCommand** بهتر است زیرا در روش استفاده از **DataCommand**فیلد هایی که **bind**نیستند مستقیما در منبع داده ای به روز رسانی می شوند و امکان انصراف برای کاربر روی این تغییرات وجود ندارد.حتی اگر کاربر تغییرات فیلد های **bind**شده را اعمال نکند ناسازگاری پیش می آید.

در پایان این مطلب را یادآور می شویم که ما از معرفی شی **DataView**که ازدیگر اشیائ مهم **ADO.Net** است چشم پوشی کردیم.در کاربرد عملی از این شی برای فیلتر کردن داده ها ومرتب سازی و جستجو در میان داده هااستفاده می شود..جستجو و فیلتر کردن را می توانید با استفاده از اشیایی که تا این جا با آن ها آشنا شدید انجام دهید.مرتب سازی را نیز می توانید از متد های معمول برای **Sort** و با کمک شی **DataRow**انجام دهید.

دستور کار جلسه 8:

در دستور کار جلسات 8و9و10تنها یک راه حل برای اعمال خواسته پیشنهاد می شود در صورت نیاز می توانید با مشورت مربی آزمایشگاه از روش های دلخواه خود برای پیاده سازی این عملیات ها استفاده کنید.

1-فرم **Main**راکدنویسی کنید.پروژه نیز تنها با بسته شدن این فرم بسته می شود.

2-کد نویسی های انجام شده پیش مطالعه را بر روی فرم **Std**اعمال کرده و سپس این فرم را به شکل زیر کامل کنید:

بعد از این که کاربر بر روی کلید"اضافه کردن دانشجوی جدید"کلیک کرد تمامی کنترل های متن های روی فرم خالی شده حال کاربر اطلاعات دانشجوی مورد نظر خود را وارد می کند.با کلیـک مجـدد

کاربر روی کلید"اضافه کردن دانشجوی جدید"اطلاعات این دانشجو مستقیما به منبع داده ای منتقل می شود.سپس **DataSet**را دوباره پر کرده و فرم ر انیز**Refresh**کنید **position**نی باید مقدار 0 را به خود بگیرد.کاری مشابه را با کلیک بر روی گزینه "حذف رکورد دانشجوی فعلی انجام دهید" با این تفاوت که این بار یک رکورد حذف می شود.می خواهیم در هنگام مرور کاربر بر روی اطلاعات دانشجویان هر تغییر داده شده بلافاصله بر روی **DataSet**اعمال شود.این کار را کد نویسی کنید سپس در صورت کلیک کاربر بر روی گزینه "اعمال تغییرات"این تغییرات بلافاصله به منبع داده ای منتقل می شود.با کلیک کاربر بر گزینه "جستجوی دانشجوفرم" جستجوی دانشجو"لود شده و با توجه به ورودی های کاربر رکورد مناسب پیدا شده و اطلاعات وی روی فرم "اطلاعات دانشجو "لود می شود.درسراسر این پروژه فرض کنید کاربر الزامی ندارد که همه فیلد های فرم های جستجو را پر کند و جستجو همیشه روی همان فیلد های مقدار دهی شده صورت می گیرد.علاوه بر این فرض کنید که در صورتی چندین رکورد مناسب با فیلد های مورد جستجو پیدا شد،همیشه رکورد اول به نمایش در می آید. در صورتی که رکوردی پیدا نشد این موضوع را به کاربر توسط یک **MessageBox**اعلام نمایید.

3-فرم های **crs**و**coPrAdd**را به شکل زیر کد نمایید:

در ابتدا و هنگام لود شدن فرم **crs**آیتم های **listBox3**رابانام دانشکده ها مقدار دهی کنید.با انتخاب یک دانشکده خاص توسط کاربر آیتم های **listBox2**با نام گروه های آموزشی آن دانشکده پر می شود.به همین نحو بعد از انتخاب گزینه مناسب از **listBox2**،آیتم های **listBox4**مقدار دهی میشوند.این کار را می توانید به هر روش دلخواه انجام دهید.با انتخاب گزینه مناسب از **listBox 4**توسط کاربر بقیه کنترل ها مقدار دهی می شوند.تکمیل لین فرم ها ر ادر جلسه بعد انجام خواهید داد.

دستور کار جلسه 9:

1- فرم های **crs**و**coPrAdd**را به شکل زیر تکمیل کنید:

در فرم **crs**با کلیک کاربر بر روی گزینه "اضافه کردن درس جدید"اتفاقی مشابه با اضافه کردن دانشجوی جدید در فرم **Std**رخ می دهد جز این که بعد از کلیک دوم کاربر بر روی گزینه "اضافه کردن درس جدید"فرم دیگری لود شده و شماره این دری در گروه آموزشی مربوطه را در خواست می کند.سپس اطلاعات این درس به منبع داده ای منتقل می شود. رویداد های "حذف رکورد فعلی درس" و اعمال تغییرات را نیز با روشی مشابه با فرم **Std**کد نویسی کنید.با کلیک کاربر بر کلید"اضافه کردن پیش نیاز یا هم نیاز فرم**coPrAdd** لود شده وبعد از روی دادن کلیک کلید"بازگشت به فرم**crs**" رکورد جدید انتخاب شده توسط کاربر را در پایگاه داده درج می کند.برای سادگی فرض می کنیم کاربرشماره درسی که می خواهدبه عنوان پیش نیاز یا هم نیاز درس جاری (رکورد جاری روی فرم**crs**)درج کند را می داند.در صورت تمایل می توانید این انتخاب را با **listBox**ها انجام دهید.

2-رویداد های فرم **trmRegistration**را کد کنید.

3- رویداد های فرم **Registration**را کد کنید.می خواهیم در این فرم بعد از انتخاب شماره دانشجو و در هنگام انتقال کنترل به **listBox1**آیتم های این **listBox**با شماره ترم های ثبت نامی دانشجو پر شود.مرتب سازی این آیتم ها را به گونه ای انجام دهید که شماره آخرین ترم دانشجو به عنوان انتخاب پیش گزیده باشد.

دستور کار جلسه 10:

1-فرم **courseMarking** را به نحو زیر برنامه نویسی کنید:
با انتخاب شماره درس و شماره ترم توسط کاربرو انتخاب گزینه "نمایش"**dataGrid**موجود در فرم با ستون های 1-نام و نام خانوادگی دانشجو 2-شماره دانشجویی 3- جنسیت 4-رشته 5-نمره از دانشجویانی که درس مذکور رل در ترم مذکور گرفته اند پر می شود .کاربر می تواند تنها در اطلاعات ستون نمره اصلاح یا درجی انجام دهد و با انتخاب گزینه به روز آوری این تغییرات در منبع داده ای اعمال می شود.
2-فرم**trmCourses**را نیز به نحو مشابه کد نویسی کنید .در این جامی خواهیم **dataGrid**شامل بر ستون های 1-نام درس 2-دانشکده درس 3-گروه آموزشی درس 4-نمره درس 5-وضعیت قبولی درس برای دانشجو در ترمی که شماره آن از کاربر گرفته شد باشد.مقدار دهی بقیه **textBox**ها را نیز با توجه به اطلاعات خواسته شده در فرم کدنویسی کنید.
3-در این مرحله کدنویسی خطاها و توجهات(**warning**)های این پروژه را انجام دهید و کاربر را از آن ها توسط **MessageBox**هایی آگاه کنید.معمول ترین این خطاها عبارتند از:1-اضافه کردن هر رکورد تکراری در هر یک از جداول (برای سادگی شما کافی است این کاررا تنها برای کلید اصلی انجام دهید یعنی تنها از درج مقادیری که کلید اصلی آن ها قبلا درج شده خودداری کنید)2-ثبت نام درس بدون رعایت پیش نیازی یا هم نیازی با ذکر شماره حداقل یکی از درس هایی که باید گذرانده شود. 3-در مورد حذف یا اصلاح رکورد های فایل های **crs**و**std**با موافقت کاربر تمامی رکورد های فایل های دیگر را که روی این دو فایل کلید خارجی دارند را با توجه به نوع تغییر حذف یا اصلاح کنید. 4-جستجوهای بدون نتیجه را به کاربر اعلام کنید.نیازی به کنترل ثبت نام بدون رعایت سقف تعداد واحد قابل اخذ در ترم نیست چون تریگری که در جلسه پنجم نوشتید این کار را انجام می دهد.