


جزوه آموزش SQL


۲.....Database	(۱)
۳.....Table	(۲)
۳.....ایجاد جدول	(۲-۱)
۳.....Schema	(۲-۲)
۳.....حذف جدول	(۲-۳)
۴.....تغییر تعریف جدول	(۲-۴)
۵.....Data Type	(۳)
۷.....Relation	(۴)
۷.....Primary Key	(۴-۱)
۷.....Foreign Key	(۴-۲)
۸.....مدیریت داده‌ها	(۵)
۸.....Insert	(۵-۱)
۸.....Select	(۲_۵)
۹.....فیلتر کردن اطلاعات (where)	(۲_۵)
۹.....مرتب کردن اطلاعات (order by)	(۵-۴)
۱۰.....Update	(۵-۵)
۱۰.....Delete	(۵-۶)
۱۰.....Truncate	(۵-۷)
۱۱.....Design Query in Editor	(۵-۸)
۱۳.....عملگر Join	(۶)
۱۳.....Cross join (الحاق)	(۶-۱)
۱۴.....Inner join (الحاق درونی)	(۶-۲)
۱۴.....Outer join (الحاق بیرونی)	(۶-۳)
۱۴.....Left join	(۶-۴)
۱۵.....Right join	(۶-۵)
۱۵.....Full outer join	(۶-۶)
۱۶.....Sub-Query	(۷)
۱۷.....کار با Apply	(۸)
۱۷.....Cross Apply	(۸-۱)
۱۷.....Outer Apply	(۸-۲)
۱۹.....ترتیب منطقی اجرا شدن قسمت‌های مختلف یک Query	(۹)

۱۹.....	عملگر ها	(۱۰)
۱۹.....	عملگر های مقایسه ای	(۱۰-۱)
۲۰.....	عملگرهای منطقی	(۱۰-۲)
۲۰.....	عملگر In و Exists	(۱۰-۳)
۲۳.....	Between	(۱۰-۴)
۲۳.....	Like	(۱۰-۵)
۲۴.....	کار با Top	(۱۰-۶)
۲۵.....	کار با Distinct	(۱۰-۷)
۲۶.....	کار با Case	(۱۰-۸)
۲۶.....	کار با IIF	(۱۰-۹)
۲۷.....	Union	(۱۰-۱۰)
۲۹.....	توابع	(۱۱)
۲۹.....	توابع تجمیعی	(۱۱-۱)
۳۰.....	توابع تاریخی	(۱۱-۲)
۳۰.....	توابع رشته ای	(۱۱-۳)
۳۲.....	ستون محاسباتی	(۱۲)
۳۲.....	ایجاد ستون های محاسباتی (Computed column)	(۱۲-۱)
۳۲.....	ایجاد ستون های مشتق شده (Derived column)	(۱۲-۲)
۳۳.....	جدول موقت	(۱۳)
۳۳.....	جدول موقت (Temp Table)	(۱۳-۱)
۳۵.....	جدول مشتق شده (Derived Table)	(۱۳-۲)
۳۶.....	Common Table Expression	(۱۳-۳)
۳۹.....	View	14)
۴۰.....	تعریف متغیر	(۱۵)
۴۳.....	Function	16)
۴۳.....	Table-Valued Function	(۱۶-۱)
۴۴.....	Scalar-Valued Function	(۱۶-۲)
۴۵.....	دستورات Rank در SQL Server	(۱۷)
۴۵.....	نحوه عملکرد دستور Ranking در SQL Server	(۱۷-۱)
۴۶.....	تابع ROW_NUMBER()	(۱۷-۲)
۴۶.....	دستور RANK در SQL Server	(۱۷-۳)
۴۷.....	دستور DENSE_RANK()	(۱۷-۴)
۴۸.....	تابع NTILE(n)	(۱۷-۵)
۵۰.....	Stored Procedures	18)
۵۲.....	کار با If	19)
۵۴.....	کار با While	(۲۰)

۵۶ Cursor	(۲۱)
۵۸ Trigger	(۲۲)
۵۸:FOR, AFTER, INSTEAD OF	(۲۲-۱)
۵۸ انواع تریگر:	(۲۲-۲)
۵۹ Deleted و Inserted جداول	(۲۲-۳)
۵۹:update(column) استفاده از تابع	(۲۲-۴)
۶۰ حذف و تغییر تریگرها:	(۲۲-۵)
۶۱ Transaction	23)
۶۴ به دام انداختن خطا (Error Trapping)	24)
۶۴ @@error	(۲۴-۱)
۶۶ Try...catch	(۲۴-۲)
۷۰ Index	(۲۵)
۷۱ Database Engine Tuning Advisor	(۲۴-۱)
۷۱ Actual Execution Plan	(۲۴-۲)
۷۴ Back up-Restore	(۲۶)
۷۵ Temporary Tables یا جداول موقت	ضمیمه A:
۷۵ Global Temp Table	جداول موقت سراسری
۷۶ Table Variable	متغیر از نوع جدول و یا

راهنمای مطالعه

تمامی دستورات قابل یادگیری در قابل قطعه کدهایی با علامت  آورده شده است.

تمامی تمرینات قابل ارائه به مدرس با نماد  آورده شده است.

نکات مهم با نماد  آورده شده است.

نکاتی که باکس‌های رنگی آورده شده است بسیار مهم است، و یادگیری آنها نیز الزامی است.

شاید برخی مثالها از جداول فرضی استفاده نمایند. جداول فرضی موجود در مثالها ممکن است با واقعیت تطابق نداشته باشد.

سوالات سخت با عبارت «سوال * دار» مشخص شده است.

متن هر سوال را بالای پاسخ خودتان به هر سوال کپی کنید تا مشخص باشد، که این پاسخ مربوط به کدام سوال است.

Database (۱)

برای ایجاد یک پایگاه داده جدید می توان از دستور زیر استفاده کرد:

CREATE Database University



و یا می توانید در قسمت Object Explorer روی Databases راست کلیک کنید و گزینه ی New Database را انتخاب کنید.

SQL Server برای دیتابیس ایجاد شده دو تا فایل تهیه می کند:

- University.MDF: اطلاعات وارد شده در این قسمت قرار می گیرند.
- University_log.LOG: اطلاعات تراکنش ها در این قسمت قرار می گیرند. فضایی هست که سرور تغییرات روی اطلاعات را اول اینجا انجام می دهد، در صورتی که موفقیت آمیز باشند بصورت اتوماتیک روی اطلاعات اصلی اعمال می شوند. این کار دارای مزایای بازبازی اطلاعات و کارآیی به ازاء هر کاربر می باشد.

Table (۲)

۲-۱) ایجاد جدول

در هر دیتابیس برای مدیریت اطلاعات از جداول استفاده می کنند. برای ایجاد جدول جدید می توانید از دستور زیر استفاده کنید:

```
CREATE TABLE [dbo].[Students](  
    [Id] [int] IDENTITY(1,1) NOT NULL,  
    [FirstName] [varchar](50) NOT NULL,  
    [LastName] [varchar](100) NOT NULL,  
    [Code] [varchar](20) NOT NULL,  
    [Address] [varchar](1000) NOT NULL,  
    [Tel] [varchar](20) NULL  
)
```



و یا می توانید در قسمت Object Explorer\Databases\University روی Tables راست کلیک کرده New Table را انتخاب کنید.

Null: به معنای صفر یا متن خالی نیست! به معنای نامشخص یا وجود نداشتن است.



مثلا شماره تلفن را در صورتی که نداند یا اصلا وجود نداشته باشد می تواند وارد نکند. اما نام حتما باید پر شود (می تواند با یک رشته مثل 'ali' و یا یک رشته خالی " پر شود).

۲-۲) Schema

Schema در واقع ساختار دیتابیس را مشخص می کند. Schema ی پیش فرض SQL ، dbo است.

ایجاد schema ی جدید:

```
Create schema testSchema
```



می توانید جداول خود را در schema ای که مد نظرتان است ایجاد کنید.

```
CREATE TABLE [testSchema].[Students] (  
    [Id] [int] IDENTITY (1,1) NOT NULL,  
    ...  
)
```



و هر جا بخواهید از این جدول استفاده کنید باید قبل از نام آن، نام schema را بنویسید (testSchema.Students). ولی اگر در schema ی dbo ایجاد کنید، مجبور نیستید نام schema را قبل از نام جدول حتما بیاورید.

۲-۳) حذف جدول

برای حذف یک جدول از دستور زیر استفاده می شود:

```
drop table students
```



و یا می توانید در Object Explorer جدول مورد نظر را انتخاب کرده راست کلیک کنید و Delete را انتخاب کنید.

۴-۲) تغییر تعریف جدول

برای تغییر تعریف جدول می توانید از دستور زیر استفاده کنید:

```
ALTER TABLE students
```

```
ALTER COLUMN Tel varchar (20) NOT NULL;
```



مثلا یک ستون null پذیر را تغییر دهیم که دیگر null قبول نکند. و یا نوع داده ای که برای یک ستون تعریف شده است را تغییر دهیم.

برای افزودن ستون جدید به جدول از دستور زیر می توانید استفاده کنید:

```
ALTER TABLE students
```

```
Add Mobile varchar (20) null
```



با اجرای این دستور ستون شماره موبایل هم به جدول دانشجویان اضافه خواهد شد.

برای حذف یک ستون از یک جدول می توانید از دستور زیر استفاده کنید:

```
ALTER TABLE students
```

```
Drop column Mobile
```



با اجرای این دستور ستون شماره موبایل از جدول دانشجویان حذف خواهد شد.

Data Type (۳)

در SQL Server هر ستون، متغیر (جلوتر بررسی خواهد شد) و پارامترها دارای نوع داده ای مشخص می باشند. طبقه بندی انواع داده در SQL Server:

فضا	محدوده	نوع داده	طبقه بندی
8 Bytes	-2^{63} to $2^{63}-1$	Bigint	عدد دقیق
8 or less bit	TRUE and FALSE	Bit	
5-9-13-17	(1-9),(10-19),(20-28),(29-38)	Decimal	
4 Bytes	-2^{31} (-2,147,483,648) to $2^{31}-1$ (2,147,483,647)	Int	
8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	Money	
5-9-13-17	نوع داده numeric شبیه به decimal است با این تفاوت که در آن استفاده از حداقل دقت و مقیاس ضروری است	Numeric	
2 Bytes	-2^{15} (-32,768) to $2^{15}-1$ (32,767)	Smallint	
4 Bytes	- 214,748.3648 to 214,748.3647	Smallmoney	
1 Byte	0 to 255	Tinyint	عدد تقریبی
8 bytes-4 bytes	15 digits-7 digits	Float	
4 Bytes	- 3.40E + 38 to -1.18E - 38, 0 and 1.18E - 38 to 3.40E + 38	Real	
3 bytes, fixed	YYYY-MM-DD	Date	
6-8 bytes	YYYY-MM-DD hh:mm:ss[.nnnnnnnn]	Datetime2	تاریخ و زمان
8 bytes	YYYY-MM-DD hh:mm:ss[.nnn]	Datetime	
10 bytes, fixed	YYYY-MM-DD hh:mm:ss[.nnnnnnnn] [+ -]hh:mm	Datetimeoffset	
4 bytes, fixed	YYYY-MM-DD hh:mm:ss	Smalldatetime	
5 bytes, fixed	hh:mm:ss[.nnnnnnnn]	Time	رشته کاراکتری
(max)2 GB Fixed-length, non-Unicode string data	from 1 through 8,000	Char	
$2^{31}-1$ (max)	Variable-length Unicode data	Text	
(max)2 GB	Variable-length, non-Unicode string data	Varchar	
(max)2 GB Fixed-length Unicode string data	from 1 through 4,000	nChar	رشته های یونیکد
$2^{30} - 1$ (max)	Variable-length non-Unicode data	nText	
(max)2 GB Variable-length Unicode string data	from 1 through 4,000	nVarchar	

n bytes	Fixed-length	binary data from 1 through 8,000	Binary
2 ³¹ -1 (max)		Variable-length binary data	Image
(max)2 ³¹ -1	Variable-length	binary data from 1 through 8,000	Varbinary

رشته
های
باینری

نکته: در صورت لزوم کاربران می توانند نوع داده های مورد نظر خود را در SQL Server تعریف کنند. البته این نوع داده ها با در نظر گرفتن شرایط خاصی برای نوع داده های تعریف شده در SQL Server شکل می گیرند. (طول مشخص یا null پذیر بودن یا نه) به عنوان مثال می توان varchar(25) را به عنوان یک نوع داده جدید تعریف کرد و در زمان تعریف جدول جدید و یا تعریف متغیر و غیره از آن بهره گرفت.



CREATE TYPE [dbo].[TBankAccNo] FROM [varchar] (25) NULL



Relation (۴)

برای درک مفهوم رابطه به دو جدول دانشجویان و رشته ها دقت کنید. هر دانشجو در یک رشته مشغول به تحصیل می باشد در نتیجه رابطه ای بین این دو جدول برقرار است که باید در دیتابیس ایجاد شود. برای ایجاد رابطه ها از ستون هایی در جداول استفاده می کنیم که دارای یک سری ویژگی ها هستند.

Primary Key (۴-۱)

به طور معمول هر جدول دارای یک کلید اصلی (PK) می باشد. یک ستون یا ترکیبی از چند ستون که در هر رکورد دارای مقدار یکتا می باشند به عنوان کلید اصلی جدول انتخاب می شوند.

نکته: می توان خصوصیت identity را برای PK فعال کرد (نوع داده ستون PK باید عدد دقیق باشد). که باعث می شود در زمان افزودن اطلاعات مقدار این ستون را خود SQL Server بصورت اتوماتیک پر کند.



Identity increment: مشخص می کند که به ازاء هر رکورد جدید عدد آن چقدر افزایش یابد.

Identity seed: اولین رکورد از چه عددی شروع شود.

نکته: مقدار ستون کلید اصلی نمی تواند null باشد.

Foreign Key (۴-۲)

یک ستون یا ترکیبی از چند ستون در یک جدول که برای برقراری ارتباط بین دو جدول ایجاد شده اند. تنها مقداری که در PK جدول اصلی قرار دارد می تواند در FK ذخیره شود، چنانچه مقداری غیر از آن وارد شود خطا خواهد داد.

برای برقراری یک ارتباط PK را از جدول اول انتخاب کرده و FK را از جدول دوم. برای این کار جدول اول را به حالت design باز کرده، راست کلیک کنید و گزینه ی Relationship را انتخاب کنید.

به عنوان مثال: برای برقراری ایجاد ارتباط بین جداول دانشجویان و رشته، ستونی در جدول دانشجویان به عنوان FK ایجاد می کنیم و آن را به ستون Id جدول رشته که کلید اصلی این جدول است و مقادیر آن یکتا است وصل می کنیم.

توجه داشته باشید که مقادیر ستون کلید خارجی می تواند تکراری باشد. مثلاً دو دانشجو در رشته کامپیوتر تحصیل می کنند.

انواع relation:



One to one: رابطه ی یک به یک، یعنی هر رکورد از جدول اول فقط می تواند به یک رکورد از جدول دوم وصل شود.

One to many: رابطه ی یک به چند، یعنی هر رکورد از جدول اول می تواند به چند رکورد از جدول دوم وصل شود.

Many to many: رابطه ی چند به چند، یعنی یک رکورد از جدول اول می تواند به چند رکورد از جدول دوم وصل شود و عکس این قضیه هم صادق است یعنی هر رکورد از جدول دوم می تواند به چند رکورد از جدول اول متصل باشد.

مثال:

۱:۱) هر گروه آموزشی فقط یک مدیر گروه دارد.

۱:N) هر استاد عضو یک دانشکده است اما هر دانشکده می تواند چند استاد داشته باشد.

N:M) دانشجو در هر ترم، چندین درس را ثبت نام می کند و یک درس به وسیله چندین دانشجو انتخاب می شود.

۵) مدیریت داده‌ها

۱-۵) Insert

برای افزودن اطلاعات به جداول می توان از این دستور استفاده کرد:

Insert into students

values ('ali','ahmadi','gandi','222222')



می توانید ستون هایی را که می خواهید مقدار دهید انتخاب کنید:

Insert into students (firstname, lastname, address)

values ('hasan', 'jalili', 'vanak')



Id کلید اصلی با خصوصیت identity است و خود SQL Server آن را مقداردهی می کند.

شماره تلفن allow null است و می توانید به آن مقدار ندهید.

۵_۲) Select

برای واکنشی اطلاعات ذخیره شده در جداول دیتابیس از این دستور می توانید استفاده کنید:

Select *

from students



* Select تمامی ستون ها را نمایش می دهد، می توان ستون هایی را که می خواهید ببینید را انتخاب کنید:

Select firstname, address

from students



دستور select را می توان بدون قسمت from هم استفاده کرد.

Select getdate()

Select 'Test'



دستور اول تاریخ و زمان لحظه ای که اجرا می شود را نشان می دهد و دستور دوم کلمه Test را در خروجی نمایش می دهد.



نکته : برای جایگزینی null در ستون های null پذیر می توان از تابع isnull استفاده کرد.

```
Select firstname, lastname, isnull ( tel, '---' ) as tel
from students
```



در دستور بالا به جای شماره تلفن هایی که وجود ندارند --- نمایش داده می شود.

۳_۵) فیلتر کردن اطلاعات (where)

برای نمایش اطلاعات دانشجویانی که اسم کوچک آنها ali است چه باید کرد؟ برای این کار از Where در دستور Select استفاده می کنیم.

```
Select * from students where firstname = 'ali'
```



می توان با And و Or شرط های دیگری هم به where اضافه کرد.

```
Select * from students where firstname = 'ali' and lastname = 'ahmadi'
```

```
Select * from students where firstname = 'ali' or lastname = 'ahmadi'
```



دستور اول : لیست دانشجویانی که نام آنها ali و فامیلی آنها ahmadi است.

دستور دوم: لیست دانشجویانی که یا نامشان ali است یا فامیلیشان ahmadi.

Ali asghari در خروجی دستور اول نمی آید اما در خروجی لیست دوم می آید. Ali ahmadi در خروجی هر دو دستور می آید.

۵-۴) مرتب کردن اطلاعات (order by)

برای مرتب کردن اطلاعات خروجی براساس یکسری ستون بصورت صعودی یا نزولی از دستور order by استفاده می شود.

```
select *
from students
where address = 'gandi'
order by firstname
```



لیست دانشجویانی که در آدرس گاندی سکونت دارند به ترتیب حروف الفبای نام آنها.

نکته : برای مرتب سازی صعودی از asc و برای مرتب سازی نزولی از desc استفاده می کنیم.



لیست دانشجویان مرتب شده براساس شماره تلفن بصورت نزولی به شکل زیر می باشد:

```
select *  
from students  
order by tel desc
```



Update (۵-۵)

برای اعمال تغییرات بر روی اطلاعات ذخیره شده در جداول از این دستور استفاده می شود. مثلا شماره ی تلفن ali اشتباه وارد شده است برای اصلاح آن به شکل زیر عمل می کنیم.

```
update students  
set tel='333333'  
where firstname='ali'
```



به صورت پیش فرض دستور Update قابل برگشت نیست، مگر اینکه داخل تراکنش اجرا شود. حتما برای تغییر Data های حساس ابتدا صحت شرط Where را چک کنید. مثلا متوانید با همان where ابتدا دستور Select نوشته و بعد از ملاحظه رکوردها (حداقل تعداد رکوردها) دستور update را اجرا کنید. اگر فیلتری بر روی اطلاعات نگذاریم، تمام اطلاعات موجود در آن ستون تغییر می کند.

Delete (۵-۶)

برای حذف یک یا چند رکورد می توان از این دستور استفاده کرد.

```
delete from students  
delete from students where firstname like 'h%'
```



دستور اول کل اطلاعات جدول دانشجویان را حذف می کند اما دستور دوم فقط رکورد هایی که اسم کوچک آنها با h شروع شده است را حذف می کند.

Truncate (۵-۷)

این دستور اطلاعات جدول را پاک می کند و log رکورد های پاک شده را نگه نمی دارد. برای جداولی که FK هایی به آنها وصل شده است نمی توان استفاده کرد. اگر PK آن جدول از نوع identity تعریف شده باشد به مقدار اولیه ی آن تنظیم خواهد شد. مثلا اگر مقدار اولیه آن یک بوده برای رکورد های جدیدی که درج شوند دوباره از یک شروع به مقدار دهی خواهد کرد. در صورتی که دستور delete این گونه رفتار نمی کند.



۸-۵) Design Query in Editor

در یک query جدید راست کلیک کنید و گزینه ی Design Query in Editor را انتخاب نمایید، به کمک این گزینه می توانید بصورت گرافیکی query بنویسید. حتی اگر قسمتی از query خود را نوشته اید و می خواهید کامل تر کنید، قسمت مورد نظر را انتخاب کنید و راست کلیک کرده و گزینه ی Design Query in Editor را انتخاب نمایید.

قوانین نگارش کد (Code Convention)

قوانین نگارش کد برای مشابه کردن دستخط افراد متفاوتی است که اقدام به نگارش دستورات SQL میکنند استفاده می شود. هر گروه برنامه نویسی خود دارای قوانین نگارش مخصوص به خود است، و یک شیوه ی نگارش الزاما درست وجود ندارد. پس لازم است با ورود به هر تیمی، قبل از اینکه اقدام به کدنویسی نمایید، با مطالعه کدهای قبلی و پرسش از برنامه نویسان قدیمی تر از قواعد نگارش آن تیم مطلع شوید.

در اینجا برای تیم آموزش یک سری قوانین نگارش در نظر گرفته شده است که در ادامه به توضیح آنها خواهیم پرداخت:

- تمامی کلمات کلیدی مانند select, from, ... با استفاده از حروف کوچک نوشته شود.
- تمامی کلمات کلیدی مانند select, from, where, having, ... در یک خط مجزا نوشته شود.
- در جدا کردن نامها که از «،» برای جدا کردن استفاده می کنید، حتما یک فاصله قرار داده شود.
- قبل و بعد از تمامی عملگرها مانند = یک جای خالی (Space) قرار داده شود.
- سعی شود که Join و عبارت On با هم در یک خط نوشته شود.
- در صورتی که یک نام از چند کلمه تشکیل شده باشد، ابتدای حروف کلمات بعدی با حروف بزرگ نوشته خواهد شد. به این شیوه نامگذاری CamelCase میگویند.
- نام مستعار برای جداول از مخفف سازی نام اصلی جدول به دست می آید.
- در نوشتن Join ها حتما Inner یا Outer بودن آن مشخص شود.

به مثال زیر برای شیوه نگارش دقت کنید.

```
select stu.FirstName + ' ' + stu.LastName as 'دانشجو نام',
       Sum ( slc.Grade * crs.Vahed ) / Sum ( crs.vahed ) as 'معدل',
       trm.title as 'ترم نام'
from Student stu
     Left Join Selection slc on stu.Id = slc.StudentId
     Inner Join Unit unt on unt.Id = slc.UnitId
     Inner Join Teach tch on tch.Id = unt.TeachId
     Inner Join Course crs on crs.Id = tch.CourseId
     Inner Join Term trm on trm.Id = unt.TermId
group by stu.Id,
         stu.FirstName,
         stu.LastName,
         trm.id,
         trm.title,
         trm.BeginDate
order by stu.Id, trm.BeginDate
```

قوانین کارایی (Performance Tips)

1. تنها ستونهای مورد نیاز را در select انتخاب کنید.
2. زمانیکه تعداد رکورد کمی داریم برای مقایسه اشکالی ندارد که از Exists و IN و SubQuery استفاده کنیم تنها باید در چنین مواقعی به جای In از Exists استفاده کنیم. ولی برای رکورد های زیاد بهتر است از Join استفاده شود.
3. حداقل امکان از Cursor استفاده نشود.
4. Having فقط زمانی استفاده شود که فیلتر ما بر روی فیلد های aggregated هست در غیر اینصورت بهتر است که از Where استفاده شود.
5. بهتر است که % ابتدای یک کلمه در سرچ نباشد چون باعث Table Scan می شود.
6. به جای Temp Table از Table Variables استفاده کنیم
7. تاجایی که امکان دارد از distinct و Union استفاده نشود و در صورت لزوم بهتر است که از Union All استفاده شود.
8. سعی شود در مواقعی که امکان نوشتن Query با Join امکان دارد؛ تا حد امکان کمتر از Sub-Query استفاده شود،

۶ عملگر Join

در مبحث مربوط به طراحی دیتابیس به این نتیجه رسیدیم که برای ایجاد یک پایگاه داده ی کارآمد باید داده ها را درون چند جدول توزیع کنیم. این کار باعث می شود هنگام بازیابی داده ها مجبور شویم داده های دو یا چند جدول را با هم ترکیب کنیم تا به نتیجه ی مورد نظر برسیم. این کار به کمک دستور Join انجام می شود.

۱-۶ Cross join (الحاق)

الحاق کردن دو یا چند جدول در SQL Server که با استفاده از دستور Join انجام می شود از یک عملیات ریاضی به نام ضرب دکارتی تبعیت می کند.

برای آشنایی با این مفهوم فرض کنید دو مجموعه زیر را داریم:

$$A = \{a1, a2, a3\}, B = \{b1, b2\}$$

حاصل ضرب دکارتی این دو مجموعه:

$$A \times B = \{ (a1,b1) , (a1,b2) , (a2,b1) , (a2,b2) , (a3,b1) , (a3,b2) \}$$

یعنی با استفاده از این عمل همه ترکیب های دوتایی از اعضای دو مجموعه ایجاد می شود به گونه ای که در زوج های مرتب تولید شده، عضو اول از مجموعه A و عضو دوم از مجموعه B است.

حال فرض کنید A و B دو جدول باشند با استفاده از دستور Cross join این دو جدول در هم ضرب دکارتی می شوند. A دارای سه رکورد و B دارای دو رکورد است و حاصل ضرب دکارتی آنها ۶ رکورد است، یعنی هر رکورد از جدول A کنار هر رکورد از جدول B قرار می گیرد.

مثال: جداول Students و SCT را در نظر بگیرید.

```
select *  
from students cross join selection
```



اسامی تمام دانشجویان را به همراه تمام اطلاعات موجود در جدول واسط students نمایش میدهد.

نکته: می توان با فیلتر کردن اطلاعات به نتایج مفیدی از این ضرب دکارتی رسید.



مثلا اگر سریال دانشجو را که در هر دو جدول وجود دارد فیلتر کنیم می توانیم اطلاعات تمامی درس هایی که هر دانشجو با اساتید مختلف اخذ کرده را به همراه نمره ی کسب کرده ی آن ببینیم.

```
select *  
from students s cross join selection se  
where s.id = se.StudentID  
order by s.id
```



در واقع می گوئیم ردیف هایی را به ما نمایش بده که در هر دو جدول مربوط به یک دانشجو هستند.

نکته : می توان برای هر ستون و یا هر جدول یک نام مستعار (alias name) برای راحتی نوشتن query انتخاب کنیم.



۶-۲ Inner join (الحاق درونی)

رایج ترین الحاق inner join است و عمل کرد آن شباهت زیادی به Cross join با where دارد. Inner join فقط ردیف هایی را برمی گرداند که در شرایط قید شده درون الحاق صدق کنند.

```
Select *  
from students s  
    inner join Selection se on s.id = se.StudentID  
order by s.id
```



خروجی این دستور همانند دستور بالا با cross join و where است.
مثال: لیست تمام دانشجویان و درس هایی که در آنها نمره ی قبولی نگرفته اند.

```
select s.FirstName + ' ' + s.LastName as [دانشجو نام],  
       c.Name as [درس نام],  
       sct.score as [نمره]  
from students s  
    inner join selection sct on s.id = sct.Student_Id  
    inner join Courses c on c.id = sct.Course_Id  
where sct.Score < 10
```



نکته : بعد از on می توان چند شرط برای تناظر رکوردها نوشت و اینکه امکان اتصال بیش از دو جدول هم وجود دارد.



۶-۳ Outer join (الحاق بیرونی)

اگر از الحاق درونی (inner join) استفاده کنیم ردیف هایی را می آورند که حداقل یک ردیف معادل در جدول دیگر داشته باشد. اگر بخواهیم اطلاعاتی را ببینیم که معادلی در جدول دیگر ندارد از outer join استفاده می کنیم.

سه نوع الحاق بیرونی وجود دارد:

Left outer join (Left join) (1)

Right outer join (Right join) (2)

Full outer join (3)

Left join (۶-۴)

در صورتی که بخواهیم از جدول اول که سمت چپ قرار دارد تمامی ردیف ها بیابند و از جدول دوم در ازاء ردیف هایی که معادلی ندارند null نمایش داده شود از left join استفاده می کنیم.

```
select *  
from students s  
    left join sct on s.id = sct.Student_Id
```



اسامی تمام دانشجویان را برمی گرداند و اگر دانشجویی درسی اخذ کرده باشد جلوی نام او مشاهده می شود و اگر بیش از یک درس اخذ کرده باشد نام او تکرار شده و در جلوی آن درسهایی که اخذ کرده نمایش داده می شوند و اگر اصلا درسی اخذ نکرده باشد جلوی نام او null نمایش داده خواهد شد.

Right join (۶-۵)

بر عکس left join عمل می کند یعنی تمام ردیف های جدول سمت راستی را نمایش می دهد و اگر معادلی از آن در سمت چپ نباشد null نمایش می دهد.

Full outer join (۶-۶)

همه رکورد ها از جداول سمت چپ و راست آورده می شوند و هر جا رکورد متناظری از جدول دیگری وجود نداشته باشد null نمایش می دهد.

تمرین



- 1- پرداخت های ترم اول دانشجوی X.
- 2- اسامی تمامی دانشجویانی که درس اخذ کرده اند به همراه نام درس و استاد.
- 3- اسامی درس هایی که تا کنون توسط هیچ استادی ارائه نشده اند.
- 4- لیست درس های اخذ شده توسط دانشجوی X که در ازای آن درس نمره ثبت شده است.
- 5- لیست دانشجویانی که در حداقل یک درس مردود شده اند.
- 6- لیست پرداخت هایی که در سال ۱۳۹۰ توسط دانشجوی X بصورت چک بوده اند.
- 7- لیست دانشجویانی که در رشته X تحصیل می کنند و در تمام درس های آن رشته نمره ی قبولی گرفته اند . (به عبارتی در آن رشته فارغ التحصیل شده اند). (سوال * دار)
- 8- اسامی تمام دانشجویان ترم یک را بیاورید، در صورتی که پرداختی داشته اند آنها هم نمایش دهید.

Sub-Query (۷)

Subquery در واقع یک query است داخل یک دستور Select، Insert، Update، Delete و یا یک subquery دیگر. یک subquery اجازه دارد در هر قسمتی از یک دستور بیايد مثلا به عنوان یک ستون و یا در قسمت های from و where.

مثال

```
select (
    select firstname
    from students s
    where s.id = sct.Student_Id
) as name,
sct.score
from sct
```



نام دانشجو را به عنوان یک ستون نمایش میدهد.

```
update Payments
set Description = (
    select firstname + ' ' + lastname
    from students s
    where s.id = Student_Id
)
where Description is null
```



توضیحات جدول پرداخت را با نام دانشجو اصلاح می کند.

```
select *
from Payments p
where p.student_id = (
    select top 1 Student_Id
    from sct
    where score = 16.5
    and sct.Year = p.Year
)
and p.year = 1393
```



پرداخت های سال ۹۳، اولین دانشجویی که در سال ۹۳ نمره ی ۱۶,۵ گرفته را نمایش می دهد.

نکته: در صورتی که subquery در قسمت From استفاده شده باشد، تمام ستون های آن حتما باید دارای نام مشخصی باشند.



تمرین



9- دستوری بنویسید که تمامی پرداخت ها به غیر از آنهایی که اسم پرداخت کننده علی است را بیاورد.

10- سه سوال مطرح کرده و از SubQuery در قسمت select، from و where استفاده کنید.

۸) کار با Apply

عمل گر Apply به شما امکان می دهد یک تابع (بعدا بررسی خواهد شد) یا یک subquery را برای هر ردیف برگردانده شده توسط جدول موجود در query فراخوانی کنید. برای به کار گیری این عمل گر، پرس وجویی ایجاد می شود که در سمت راست APPLY یک تابع یا subquery قرار دارد و در سمت چپ آن یک جدول قرار گرفته است؛ به ازای هر ردیفی که از جدول استخراج می شود، مقدار تابع یا subquery ارزیابی شده و برای تولید خروجی نهایی مورد استفاده قرار می گیرد. مجموعه ستون هایی که در خروجی نهایی دیده می شوند، شامل ستون های جدول و نیز ستون های برگردانده شده توسط تابع یا subquery هستند. از دید منطقی دستور سمت راست به ازای هر رکورد یک بار اجرا میشود. البته ممکن هست به علت Optimization عملا این اتفاق رخ ندهد. عملگر apply به دو صورت زیر مورد استفاده قرار می گیرند:

۸-۱) Cross Apply

این عملگر فقط ردیف هایی از جدول را برمی گرداند که مجموعه نتایجی را برای تابع inline یا subquery تولید می کنند.

۸-۲) Outer Apply

ای عملگر همه ردیف های جدول را بر می گرداند، چه آنهایی که منجر به برگشت نتیجه از تابع یا subquery می شوند و چه آنهایی که برای تابع یا Subquery، خروجی تولید نمی کنند. درحالت دوم به جای خروجی تابع یا subquery مقدار null بر می گرداند.

به عنوان مثال تابعی را در نظر بگیرید که با گرفتن سریال دانشجویی معدل دانشجو را در هر ترم بر می گرداند.

```
Select s.firstname, tb.GPA, tb term
from students s cross apply
(
    select avg ( score ) GPA, term
    from sct
    where Student_Id = s.id
    group by term
) tb
```



```
select s.firstname, tb.GPA, tb.term
from students s outer apply
(
    select avg ( score ) GPA, term
    from sct
```

```

where Student_Id = s.id
group by term
) tb

```

نتیجه select اول

<u>Firstname</u>	<u>GPA</u>	<u>term</u>
Ali	17.50	1
Hassan	13.33	1

نتیجه select دوم

<u>Firstname</u>	<u>GPA</u>	<u>term</u>
Ali	17.50	1
Hassan	13.33	1
Mina	null	null

برای mina هیچ رکوردی در جدول sct درج نشده است.

نکته: توابعی که ورودی آنها از ستون های جدول موجود در قسمت from استفاده می کنند، باید به کمک دستور Apply در query بکار روند.



نکته: تابعی که ورودی ندارد می تواند با استفاده از cross join نیز در قسمت From استفاده شود.



نکته: تابعی که ورودی های آن به شکل متغیر در پروسیجر وجود دارند نیازی به استفاده از apply ندارند.



تمرین

11- دستوری بنویسید که پرداخت های آخرین دانشجویی که در ترم یک نمره قبولی گرفته است را نمایش دهد. (سوال * دار)



۹) ترتیب منطقی اجرا شدن قسمت‌های مختلف یک Query

ترتیب منطقی اجرا شدن قسمت‌های مختلف یک دستور select به شکل زیر است: (بسیار مهم)

1. FROM
2. ON
3. JOIN
4. WHERE
5. GROUP BY
6. HAVING
7. DISTINCT
8. ORDER BY
9. TOP
10. SELECT

۱۰) عملگرها

برای انجام یکسری از محاسبات و بدست آوردن نتایج مورد نظر می‌توانیم از عملگرهای مختلف موجود در SQL استفاده کنیم.

۱-۱۰) عملگرهای مقایسه‌ای

عملگر	کارکرد	مقایسه
=	مساوی با	'ali' =
>	بزرگتر از	> 20
<	کوچکتر از	< 20
=>	بزرگتر یا مساوی	=> 20
<=	کوچکتر یا مساوی	=< 20
<>	نامساوی (مخالف)	<> 20

مثال: اسامی دانشجویانی که در رشته کامپیوتر درس نمی‌خوانند براساس حروف الفبای فامیلی آنها.

```
select s.firstname, s.lastname, s.code, r.name
from students s
inner join reshte r on s.reshte_id = r.id
```



```
where r.name <> 'computer'
order by s.lastname
```

تمرین



- 12- لیست دانشجویانی که حداقل یک نمره بین ۲۰ و ۱۵ است.
13- لیست دانشجویانی که تاکنون ۲۰ نگرفته اند.

۲-۱۰) عملگرهای منطقی

And: مانند "و" منطقی عمل می کند یعنی باید همه شرط ها درست باشند تا کل عبارت درست باشد.
Or: مانند "یا" منطقی عمل می کند یعنی درست بودن یک شرط برای صحیح بودن کل عبارت کافی است.

تمرین



- 14- لیست دانشجویانی که حداقل بخشی از شهریه خود را یا بصورت فیش الکترونیکی و یا دستگاه Pos داده اند.

۳-۱۰) Exists و In عملگر

با روش بررسی مقادیر موجود در پایگاه داده با استفاده از Or آشنا شدید. مثلا اسامی دانشجویانی که یا در رشته کامپیوتر یا برق یا عمران تحصیل می کنند.

```
Select s.firstname, s.lastname, s.code, r.name
```

```
from students s
```

```
inner join reshte r on s.reshte_id = r.id
```

```
where r.name = 'computer' or r.name = 'bargh' or r.name = 'omran'
```



عملگر in تا این query را راحت تر بنویسید.

```
select s.firstname, s.lastname, s.code, r.name
```

```
from students s
```

```
inner join reshte r on s.reshte_id = r.id
```

```
where r.name in ( 'computer', 'bargh', 'omran' )
```



نکته : از دستور In زمانی استفاده می شود که قرار است فیلد مورد نظر را با تعداد محدودی از اطلاعات مقایسه کرد اگر از تعداد زیادی استفاده کنیم کارایی به خطر می افتد و آن موقع بهتر است از دستور Exists استفاده کنیم.



```
select name
from reshte r
where id in (
    select reshte_id
    from students
)
```



```
select r.name
from reshte r
where exists (
    select s.reshte_id
    from students s
    where r.id = s.reshte_Id
)
```

زمانیکه حجم اطلاعات بالا است و تعداد رکورد های برگشتی select در قسمت where بالا باشد دستور دوم کارایی بهتری خواهد داشت.

مثال زیر تفاوت In و Exists را برای مقادیر null نمایش می دهد.

```
CREATE TABLE t1 ( id INT, title VARCHAR ( 20 ) , someIntCol INT )
GO
CREATE TABLE t2 ( id INT, t1Id INT, someData VARCHAR ( 20 ) )
GO
```



```
INSERT INTO t1
SELECT 1, 'title 1', 5 UNION ALL
SELECT 2, 'title 2', 5 UNION ALL
SELECT 3, 'title 3', 5 UNION ALL
SELECT 4, 'title 4', 5 UNION ALL
SELECT null, 'title 5', 5 UNION ALL
SELECT null, 'title 6', 5
```

```
INSERT INTO t2
SELECT 1, 1, 'data 1' UNION ALL
```



```

SELECT 2, 1, 'data 2' UNION ALL
SELECT 3, 2, 'data 3' UNION ALL
SELECT 4, 3, 'data 4' UNION ALL
SELECT 5, 3, 'data 5' UNION ALL
SELECT 6, 3, 'data 6' UNION ALL
SELECT 7, 4, 'data 7' UNION ALL
SELECT 8, null, 'data 8' UNION ALL
SELECT 9, 6, 'data 9' UNION ALL
SELECT 10, 6, 'data 10' UNION ALL
SELECT 11, 8, 'data 11'

```

```

SELECT *
FROM t1
WHERE not EXISTS (
    SELECT *
    FROM t2
    WHERE t1.id = t2.t1id
)

```

```

SELECT t1.*
FROM t1
WHERE t1.id not in (
    SELECT t2.t1id
    FROM t2
)

```

نتیجه select اول (Exists)

Id	title	someIntCol
	NULL	title 5 5
	NULL	title 6 5

نتیجه select دوم (in)

Id	title	someIntCol
----	-------	------------

نکته: not exists و not in هم در واقع معکوس in و exists عمل می کنند.

تمرین



15- لیست اساتیدی که دروس معماری و ساختمان داده را با هم در یک ترم ارائه داده اند.
(سوال * دار)

16- لیست نوع پرداخت هایی که تا کنون دانشجویی از آنها استفاده نکرده است.

۴-۱۰) عملگر Between

برای برگرداندن اطلاعاتی که در یک محدوده خاص قرار دارند. مثلا برگرداندن اطلاعاتی که در یک بازه زمانی مشخص اتفاق افتاده اند.

لیست دانشجویانی که نمره هایی بین ۲۰ و ۱۵ گرفته اند.

```
select s.FirstName + ' ' + s.LastName as [دانشجو نام], sct.score as [نمره]
from students s
      inner join sct on s.id = sct.Student_Id
where sct.Score between 15 and 20
```



۵-۱۰) عملگر Like

این عملگر ابزاری کارآمد برای جستجو در میان فیلدها و پیدا کردن موارد مشابه با عبارت مورد نظر است.

Name like 'ali' لیست افرادی که نام آنها ali می باشد.

Name like 'a%' لیست اسامی که با a شروع می شوند طول آنها مهم نمی باشد.

Name like '%a%' لیست افرادی که در نام آنها حرف a وجود دارد.

Name like 'a__' لیست افرادی را می آورد که اسم آنها سه حرفی است و با حرف a شروع می شود. (تعداد حروف در اینجا مهم است و با _ به اِزاء هر حرف مشخص می شود)

Name like 'a[a-z][0-9]' لیست افرادی را می آورد که اسمشان با حرف a شروع می شود و دومین حرف حتما از حروف الفبا است و حرف سوم یک عدد بین ۰ تا ۹

نکته: اگر متن جستجو از نوع nvarchar(max) باشد نمی تواند طول بیشتر از ۴۰۰۰ داشته باشد.



```
declare @cmnd nvarchar ( max ) = replicate ( 'a', 4001 )
select 'Ali Mohammadi'
where 'Ali Mohammadi' like @cmnd
```



نتیجه

Msg 8152, Level 16, State 10, Line 2
String or binary data would be truncated.

اما در متن جستجو که در سمت چپ این عملگر قرار می گیرد این محدودیت وجود ندارد.

```
declare @cmnd nvarchar ( max ) = replicate ( 'ali mohammadi ', 8000 )
```

```
select 'Ali Mohammadi'
where @cmnd like 'Ali Mohammadi%'
```



نتیجه

'Ali Mohammadi'

تمرین



17- لیست پرداخت هایی که در قسمت توضیحات آنها از کلمه ی 'مشکل' یا 'ایراد' و یا '%' استفاده شده باشد.

۶-۱۰ کار با Top

گاهی لازم است در هنگام کار با دستور select تنها بخشی از داده ها که در شرایط می گنجد را مشاهده کنید. برای مثال هنگامی که در یکی از وب سایت های جستجو مانند Google، عبارتی را جستجو می کنید ممکن است هزاران نتیجه برای نمایش وجود داشته باشد اما نمایش یکباره آنها ضرورت ندارد چون صرف نظر از مشکلات فنی مانند مشغول شدن بیش از حد پایگاه داده، ممکن است نتیجه مورد نظر کاربر در میان چند گزینه اول باشد.

برای نمایش بخشی از اطلاعات از top در کنار select استفاده می شود که قادر است تعداد یا درصد مشخصی از رکوردها را برگرداند.

```
select top 5 * from students
select top 5 percent * from Students
```



دستور اول تعداد ۵ دانشجوی اول را می آورد و دستور دوم ۵٪ کل دانشجویان را می آورد.

نکته: می توان از with tie به همراه top استفاده کرد که در این صورت باید حتما از order by هم استفاده شود.



```
select top 2 with ties Id
from students
order by firstname
```



نتیجه

Id	FirstName	LastName
۱	ali	ahmadi

۳	hasan	jalili
۹	hasan	gholami

همانطور که از نتیجه مشخص است با اینکه گفته شده Top 2 اما سه ردیف در خروجی نمایش داده شده است، علت آن استفاده از دستور with ties می باشد. در این query ابتدا اطلاعات مرتب می شوند و بعد به دستور top n نگاه می کنند، n ردیف اول را جزء خروجی می آورد و اگر در ستونی که در order by آمده اطلاعات مشابهی با ردیف n ام وجود داشته باشد آنها را هم به خروجی اضافه می کند.

تمرین



18- لیست ۱۰ درصد آخرین دانشجویانی که انتخاب واحد کرده و نمره گرفته اند.

۷-۱۰ کار با Distinct

در هنگام کار با دستور select از این عبارت برای نمایش ردیف های غیر تکراری استفاده می شود. در زمان استفاده از Distinct باید ستون ها بادقت بیشتری انتخاب شوند.

به عنوان مثال فکر کنید اسامی دانشجویانی که حداقل یک نمره بالاتر از ۱۵ دارند را می خواهیم بدانیم.

```
select s.firstname, s.lastname
from students s
inner join sct on s.id = sct.Student_Id
where sct.score > 15
```



با این دستور اگر ali دو تا نمره بالاتر از ۱۵ داشته باشد در دو ردیف نمایش داده می شود و همینطور بقیه دانشجویان.

```
select distinct firstname, lastname
from students s
inner join sct on s.id = sct.Student_Id
where sct.score > 15
```



اما با استفاده از distinct بعد از select ردیفهای تکراری نمایش داده نمی شوند.

تمرین



19- لیست نوع پرداخت هایی که مبلغ بالاتر از ۵۰۰۰۰ تومان هم داشته اند.

۸-۱۰ کار با Case

تابع Case یک مقدار را با چند شرط تطبیق می دهد و بسته به حالت هایی که برای آن تعریف شده است، مقدار خاصی را برمی گرداند. Case بر خلاف If به تنهایی کاربرد ندارد، بلکه بخشی از یک عبارت Select یا Update محسوب می شود.

مثال: به ازاء هر تاریخ ورودی اسم ماه را بنویسد.

```
select
  case substring ( '93/03/04', 4, 2 )
  when '01' then 'فروردین'
  when '02' then 'اردیبهشت'
  when '03' then 'خرداد'
  else 'ها ماه سایر'
end
```



از تاریخ داده شده ماه را به کمک substring استخراج می کند و از بالا تک تک با شرط ها مقایسه می کند و اگر در هیچکدام صادق نباشد به قسمت else می رود.

می توان در قسمت شرط و نتیجه از عملیات پیچیده تری استفاده کرد.

```
select
  case
  when score < 10 then 'ضعیف'
  when score between 10 and 15 then 'متوسط'
  when score > 15 then 'است ممتاز دانشجوی' + ' ' + (
    select firstname
    from students s
    where sct.Student_Id = s.Id
  )
end
from sct
```



۹-۱۰ کار با IIF

تابع IIF() مشابه تابع CASE() عمل می کند با این تفاوت که CASE در تمام نسخه های SQL قابل اجرا است ولی IIF فقط در SQL2012 به بعد استفاده می شود.

```
IIF ( Value1 = Value2, 'trueValue', 'falseValue' )
```

زمانی که شرط برقرار باشد مقدار trueValue برگردانده میشود و در غیر این صورت مقدار falseValue.

نکته : می توانیم از IIF های تودرتو استفاده کنیم .



نکته: می توانیم از چند شرط در این تابع استفاده کنیم.



```
SELECT IIF ( substring ( '93/03/04', 4, 2 ) = '03' and  
            substring ( '93/03/04', 7, 2 ) = '04', 'خرداد', '0' ) AS [month]
```

تمرین



20- لیست دانشجویان و درسهایی که در ترم یک اخذ کرده اند را بیاورید و فعال یا غیر فعال بودن دانشجو و عملی یا غیرعملی بودن را بصورت رشته مشخص نمایید و همچنین به ازاء نمرات بالای ۱۷ حرف A و بین ۱۵ تا ۱۷ حرف B و بین ۱۲ تا ۱۵ حرف C و کمتر از ۱۲ حرف D را نمایش دهید.

۱۰-۱۰ عملگر Union

عملگر Union (اجتماع) به شما امکان می دهد مجموعه نتایج حاصل از اجرای دو یا چند select را با هم ترکیب نموده و یک مجموعه واحد ایجاد کنید. مشروط به اینکه نوع داده ای و تعداد ستون ها با هم تطابق داشته باشند.

```
select firstname, lastname  
from students s  
join sct on s.id = sct.student_id  
where sct.course_id = 1
```

union

```
select firstname, lastname  
from students s  
join sct on s.id = sct.student_id  
where sct.course_id = 2
```



نکته: دستور union خروجی بالا و پایین را با حذف رکورد های تکراری نمایش می دهد. اگر می خواهید رکورد های تکراری را هم ببینید از union all استفاده کنید.



نکته: نام ستون ها از روی select اول برداشته می شود.



نکته : در مقایسه union با join به این نتیجه می رسیم که join جداول را بصورت افقی (تعداد ستون ها) با هم ترکیب می کند اما union به صورت عمودی (تعداد ردیف ها).



تمرین



21- کارنامه درسی دانشجو X در ترم یک (لیست درس و نمره و ردیف مجموع که معدل را حساب کرده است،) برای مثال: (سوال * دار)

۱۶	ریاضی ۱
۱۳	تاریخ معاصر
۱۸	فیزیک ۱
۱۶	معدل کل

22- کارنامه مالی دانشجو X در ترم یک (لیست درسها و هزینه هر درس و شهریه ثابت و مجموع شهریه ترم- نمایش پرداخت های داشجو - نمایش مبلغ بدهکاری/ بستانکاری دانشجو (BdBs) بدون در نظر گرفتن ترم محاسبه شود و مبلغ تسویه شده و تسویه نشده و پرداختی استفاده نشده) (سوال * دار)

(۱۱) توابع

۱-۱۱) توابع تجمیعی

گاهی اوقات لازم می شود برحسب مقادیر در یک فیلد، محاسبه ای روی فیلد دیگری انجام دهید. مثلا بیشترین نمره به ازاء هر دانشجو، یا مجموع نمرات هر دانشجو. در این گونه موارد توابع تجمیعی به ما کمک می کنند.

تابع	عملکرد
Min	محاسبه کوچکترین مقدار
Max	محاسبه بزرگترین مقدار
Sum	محاسبه مجموع
Avg	محاسبه میانگین
Count	شمارش تعداد

نکته: برای بدست آوردن بیشترین نمره **به ازاء** هر دانشجو علاوه بر Max به **Group by** نیاز داریم. تا مفهوم "به ازاء" را پیاده سازی کنیم.



```
select max ( score )
from sct
group by student_id
```



اگر group by را حذف کنید بیشترین نمره در کل جدول را می دهد.

نکته : زمانیکه از Group by استفاده می کنید در دستور Select برای اعمال فیلتر روی مشخصات گروه باید از **Having** استفاده کنید.



مثلا بدست آوردن بیشترین نمره به ازاء هر دانشجو که حداقل دو درس اخذ کرده است.

```
select max ( score )
from sct
group by student_id
having count ( course_id ) > = 2
```



تمرین



- 23- معدل هر دانشجو را در هر ترم محاسبه کنید.
- 24- اسامی دانشجویانی که کمتر از ۳ درس اخذ کرده اند را اعلام کنید.
- 25- بیشترین تعداد واحد اخذ شده در کدام ترم است و توسط چه کسانی اخذ شده است. (نمایش فقط برای یک ترم) (سوال * دار)

۲-۱۱) توابع تاریخی

عملکرد	تابع
تاریخ جاری به میلادی	Getdate()
روز را از تاریخ ورودی بر می گرداند	Day(Getdate())
شماره ماه را بر می گرداند	Month()
سال را بر می گرداند	Year()
اختلاف دو تاریخ ورودی را با توجه به interval تعیین شده بر می گرداند. Interval می تواند dd, mm, yy باشد.	Datediff(interval,date1,date2)

```
select getdate(),day(getdate()),month(getdate())
```

```
select DATEDIFF ( dd, '2014-10-20 15:09:52.130', '2014-10-18 15:10:25.253' )
```



تمرین

26- چند روز به پایان امسال میلادی باقی مانده است؟



۳-۱۱) توابع رشته ای

بخش عمده ی اطلاعات بصورت رشته ای در دیتابیس ذخیره می شوند مثلاً نام، نام خانوادگی، آدرس و ... برای کار با رشته ها توابع مختلفی در دیتابیس وجود دارد.

مثال	عملکرد	تابع
Len('abcde')=5	طول رشته را محاسبه می کند.	Len(string)
Left('abcde',2)='ab'	از سمت چپ رشته a حرف بر می گرداند.	Left(string,a)
Substring('abcde',3,2)='cd'	از حرف sام رشته a حرف بر می گرداند.	Substring(string,s,a)
Right('abcde',2)='de'	از سمت راست رشته a حرف را برمیگرداند.	Right(string,a)

مثال اسامی دانشجویانی که کد دانشجویی آنها با ۹۳ شروع شده است.

```
select *, left ( code, 2 )  
from students  
where left ( code, 2 ) = '93'
```



تمرین



27- رشته '93/07/26' را بصورت سال، ماه و روز جدا جدا نمایش دهید.

28- سوال بالا را با استفاده از مفهوم Cast کردن به فرمت Date و با استفاده از توابع Year، ... بنویسید. (برای آشنایی با روش Cast کردن و استفاده از توابع مربوط به تاریخ از اینترنت کمک بگیرید)

نکته : با استفاده از عملگر + می توانید دو رشته را به هم بچسبانید.



```
select code + '_' + firstname + ',' + lastname  
from students  
order by lastname,firstname desc
```



تمرین



29- تمامی شهریه ثابت تسویه شده را به شکل زیر نمایش دهید. (با توجه به نکته زیر)

مبلغ بابت شهریه ترم..... توسط دانشجوی پرداخت شده است.

نکته: یک نوع عددی را نمی توان به یک رشته متصل کرد باید نوع آن را عوض کنیم. برای این کار از دستور Cast استفاده می کنیم.



```
Select cast ( 12.50 as varchar (5) ) decimalToString,  
Cast ( '12.50' as decimal (18,2) ) stringToDecimal
```



ستون محاسباتی (۱۲)

۱۲-۱) ایجاد ستون های محاسباتی (Computed column)

برای ایجاد چنین فیلدهایی جدول را به حالت design باز کرده فیلد جدیدی اضافه می کنیم نوع آن را متناسب با خروجی محاسباتمان قرار می دهیم و در قسمت `computed column specification\formula` محاسبات مورد نظر را می نویسیم.

مثلا اگر بخواهیم تمامی نمره ها به اضافه ۲ نمره، را در ستون جداگانه ای داشته باشیم می توانیم در جدول SCT فیلد جدید را اضافه کرده و `score+2` را در قسمت `formula` بنویسیم.

تمرین

30- در جدول Selection ماه زمان ثبت نمره را در یک ستون جدا محاسبه کنید.



۱۲-۲) ایجاد ستون های مشتق شده (Derived column)

ستون مشتق شده در واقع یک Subquery است که یک مقدار برمی گرداند. این Subquery می تواند به query بیرونی وابسته باشد و یا نباشد. اما خروجی آن باید دقیقا یک ستون و یک ردیف باشد .

```
select (  
    select firstname  
    from students  
    where id = sct.student_id  
    ) name, score  
from sct
```



تمرین

31- لیست تمامی دانشجویان و تعداد درسهای که اخذ کرده اند را به کمک `derived column` نمایش دهید.



جدول موقت (۱۳)

۱-۱۳ جدول موقت (Temp Table)

گاهی اوقات با توجه به پیچیده بودن عملیات از جداول موقتی برای کار با اطلاعات استفاده می شود ، مجموعه ای از نتایج را درون یک جدول موقت قرار داده و سپس عملیات مورد نظر را روی آن انجام می دهند. مراحل استفاده از این جداول به این صورت می باشد: ایجاد جدول موقت، وارد کردن داده ها درون آن، بازیابی نتایج مورد نظر و نهایتاً حذف جدول موقت.

```
create table #studentTemp (  
    student_id int,  
    term_id int,  
    moadel  
    float  
)
```



```
insert into #studentTemp  
select  
  
    , term, sum (score * unit) / sum (unit) moadel  
from sct  
    inner join Courses c on sct.Course_Id = c.Id  
group by student_id, term
```

```
update payments  
set description = (  
    select " + cast ( moadel as varchar ( 10 ) )  
    from #studenttemp t  
    where payments.student_id = t.Student_Id and  
        Payments.Term=t.Term  
)
```

```
drop table #studenttemp
```

راه آسان تر برای ایجاد جدول موقت در این مثال به شکل زیر است:

```
select student_id, term, sum ( score * unit ) / sum ( unit ) moadel  
into #studenttemp  
from sct  
    inner join Courses c on sct.Course_Id = c.Id  
group by student_id, term
```



استفاده از دستور *into* که خودش جدول موقت را ایجاد می کند و اطلاعات خروجی از select هم در آن قرار می دهد. اما بعضی مواقع مثلا زمان ریختن خروجی یک پروسیجر (بعدا آشنا می شویم) باید دستور Create جدول موقت را نوشت.

نکات:

- این نوع جدول تنها در طول حیات یک session (صفحه کویری باز شده) و یا procedure ی که آنرا ایجاد کرده قابل دسترسی است.
- با پایان session و یا اتمام اجرای procedure ایجاد کننده جدول بطور اتوماتیک حذف خواهد شد.
- قابلیت اعمال دسترسی برای این نوع جدول وجود ندارد.
- جدول ایجاد شده در یک procedure می تواند در procedure های صدا شده توسط آن procedure استفاده شود ولی عکس آن صادق نمی باشد.
- جدول ایجاد شده در procedure و یا trigger فراخوانی شده می تواند هم نام جدول ایجاد شده قبل از فراخوانی procedure و یا trigger باشد. در procedure های تودرتو نیز می توان جدول همانم جدول ایجاد شده قبل از فراخوانی آن داشت، با این تفاوت که باید ساختار جدول همانم، یکی باشند. به مثال زیر توجه کنید و سعی کنید ساختار جدول موقت را تغییر داده و نتیجه را ببینید.

```
CREATE PROCEDURE dbo.Test2
```

```
AS
```

```
CREATE TABLE #t ( x INT PRIMARY KEY );
```

```
INSERT INTO #t VALUES ( 2 );
```

```
SELECT Test2Col = x FROM #t;
```

```
GO
```

```
CREATE PROCEDURE dbo.Test1
```

```
AS
```

```
CREATE TABLE #t ( x INT PRIMARY KEY );
```

```
INSERT INTO #t VALUES ( 1 );
```

```
SELECT Test1Col = x FROM #t;
```

```
EXEC Test2;
```

```
GO
```

```
CREATE TABLE #t ( x INT PRIMARY KEY );
```

```
INSERT INTO #t VALUES ( 99 );
```

```
GO
```

```
EXEC Test1;
```

```
GO
```

```
----- result -----
```

```
(1 row(s) affected)
```

```
Test1Col
```

```
-----
```



1

(1 row(s) affected)

Test2Col

2

جدول موقتی که تا کنون معرفی شد با یک علامت # را جدول موقت محلی (#TempTable) می باشد.

جدول موقت دیگری هم در SQL وجود دارد که با دو علامت # مشخص می شود و جدول موقت سر/سری (##TempTable) نام دارد، چرا که این نوع جدول پس از ایجاد آن می تواند توسط session های دیگر استفاده شوند. با پایان session ایجاد کننده و آخرین transaction استفاده کننده از آن بطور اتوماتیک حذف خواهد شد.

تشخیص وجود جدول موقت:

```
select object_id ( 'TempDB.dbo.#T1' )
```



در صورت وجود جدول موقت #T1 سریال آن برمیگردد در غیر این صورت Null به عنوان خروجی نمایش داده می شود.

۲-۱۳) جدول مشتق شده (Derived Table)

جدول مشتق در مقابل جدول موقت محبوبیت بیشتری دارد، چرا که جدول مشتق شده به سادگی و در یک مرحله مورد استفاده قرار می گیرد و ضمناً سرعت اجرای بالاتری دارد اما جدول موقت تعداد مراحل بیشتری در زمان استفاده دارد. جدول مشتق شده در واقع بصورت sub query در قسمت From می باشد.

مثال : می خواهیم تعداد درس های اخذ شده توسط دانشجویان را بررسی و دسته بندی کنیم و در ادامه تعیین کنیم که در هر دسته چند دانشجو وجود دارد.

دسته	تعداد درسهای اخذ شده
A	بیشتر از ۳۰ درس
B	بین ۱۰ تا ۳۰ درس
C	کمتر از ۱۰ درس

در ابتدا باید مشخص کنیم که هر دانشجو چند درس اخذ کرده است:

```
select Student_Id, count ( distinct course_Id ) course_cnt
```

```
from sct
```

```
group by Student_Id
```



حال با افزودن Case دسته را مشخص می کنیم:

```
select Student_Id, count ( distinct course_Id ) course_cnt,  
       case  
       when count ( distinct course_Id ) > 30 then 'A'  
       when count ( distinct course_Id ) between 30 and 10 then 'B'  
       when count ( distinct course_Id ) < 10 then 'C'  
       end as level  
from sct  
group by Student_Id
```



حال می خواهیم تعداد دانشجویان در هر دسته را مشخص کنیم:

```
select count ( student_id ) student_cnt, level  
from  
  (  
    select Student_Id, count ( distinct course_Id ) course_cnt,  
           case  
           when count ( distinct course_Id ) > 30 then 'A'  
           when count ( distinct course_Id ) between 30 and 10 then 'B'  
           when count ( distinct course_Id ) < 10 then 'C'  
           end as level  
    from sct  
    group by Student_Id  
  ) tbl  
group by level
```



Common Table Expression (۱۳-۳)

یک CTE را می توانید مجموعه موقت از نتایج تصور کنید. یک CTE از این منظر که بصورت یک شی در پایگاه داده ذخیره نمی شود و طول عمر آن محدود به زمان آن query است مانند جدول مشتق شده است اما بر خلاف Derived Table می تواند به خود ارجاع داشته باشد (self join).

```
with cte as(
    select *
    from edu.student
)

select *
from cte a
inner join cte a1 on a.id=a1.id
```

در صورت اجرای کویری زیر خطا رخ می دهد:

```
select *
from ( select *
      from edu.student
    ) a
inner join a a1 on a.id=a1.id
```

پیغام خطا: Invalid object name 'a'.

استفاده از CTE مزیت هایی دارد از جمله آن ها افزایش خوانایی و سادگی نگهداشت query پیچیده است. در این حالت query به چند قطعه ساده و مجزا تبدیل می شود. گاهی مواقع می توان به جای ساخت View از CTE استفاده کرد. نحوه استفاده از آن به شکل زیر است:

```
with cte_student as
(
    select distinct student_id, teacher_id
    from sct
    where sct.teacher_id = 1 and score < 20
)

select *
from cte_student
```



برای آشنایی بیشتر با جداول موقت به ضمیمه A از همین جزوه مراجعه شود.



برای جلوگیری از ایجاد خطا قبل از هر CTE از علامت ; و یا go استفاده شود.



تمرین



32- میزان بدهی تسویه نشده دانشجویان ترم یک را محاسبه کنید. برای این کار به سه صورت متفاوت و هر بار با استفاده از یکی از موارد زیر مساله را حل نمایید. (و execution Plan هر سه جواب را با هم مقایسه کنید.) (سوال * دار)

a. جدول موقت

b. جدول مشتق

c. CTE

View (۱۴)

View که نما یا دیدگاه هم گفته می شود، یک جدول مجازی است که در قالب یک query در پایگاه داده ذخیره می شود و داده های یک و یا چند جدول را به شکلی قابل فهم در معرض دید کاربر پایگاه داده قرار می دهد. توجه داشته باشید که درون view هیچ داده ای وجود ندارد و فقط query سازنده view در پایگاه داده ذخیره شده و هنگام باز شدن نما، داده ها را از جداول استخراج می کند و نمایش می دهد. ایجاد نما در یک پایگاه داده غالباً به دو دلیل زیر انجام می پذیرد:

افزایش امنیت: گاهی اوقات مشاهده اطلاعات همه ی سطر ها یا ستون های یک جدول توسط کاربر مجاز نیست. در این حالت نمایی از جدول ساخته می شود که فاقد ستون ها یا سطرهای دارای اطلاعات حساس است.

افزایش خوانایی داده ها: در بخش مربوط به طراحی پایگاه داده مشاهده نمودید که داده ها باید در چند جدول توزیع شوند تا بتوان ذخیره و بازیابی اطلاعات را با سرعت و صحت مناسبی انجام داد. از این رو در تعدادی از جداول، مقادیر برخی ستون ها به صورت ارجاعی نگهداری می شوند. اگر بخواهید داده ها را در معرض دید یک کاربر معمولی قرار دهید یا آن ها را در گزارشی نمایش دهید باید این مقادیر ارجاعی را به مقدار قابل فهم تبدیل نمایید.

مثلاً در جدول SCT سریال دانشجوی و درس و استاد وجود دارد که برای کاربر معمولی قابل فهم نیست. می توان یک View با query زیر داشت:

```
CREATE VIEW [dbo].[View_1]
```

```
AS
```

```
SELECT s.Code, s.FirstName + ' ' + s.LastName AS studentName, c.Name AS  
courseName, t.FirstName + ' ' + t.LastName AS teacherName, dbo.SCT.Score
```

```
FROM dbo.SCT
```

```
INNER JOIN dbo.Students AS s ON s.Id = dbo.SCT.Student_Id
```

```
INNER JOIN dbo.Courses AS c ON c.Id = dbo.SCT.Course_Id
```

```
INNER JOIN dbo.Teachers AS t ON t.Id = dbo.SCT.Teacher_Id
```



برای ایجاد یک View به صورت گرافیکی می توانید با راست کلیک روی object explorer\ databases\ university\ views گزینه New View را انتخاب کنید.

نکته : می توان عملیات محاسباتی هم در view انجام دهید.



تمرین

33- یک View بسازید که لیست دانشجویان به همراه میزان مبلغ تسویه نشده و پرداختی استفاده نشده آنها را در هر ترم نمایش دهد.



(۱۵) تعریف متغیر

برای تعریف متغیر در SQL Server به شکل زیر عمل می کنیم:

```
declare @name [data type]
```



```
declare @id int
```

```
declare @name varchar ( 100 ) = "
```

```
declare @score real = 0.0
```

```
select @id as id, @name as name, @score as score
```

برای متغیرهای @name و @score مقادیر پیش فرض تعیین شده است. در نتیجه اگر با مقداری پر نشوند مقدار پیش فرض را نمایش خواهند داد.

برای مقدار دهی متغیرها می توان از دو دستور set و select استفاده کرد.

```
set @name = 'ali'
```

```
set @score = 19.5
```



```
select @name = 'ali', @score = 19.5
```

```
select top 1 @name = firstname, @score = score  
from sct
```

```
inner join students s on sct.student_id = s.id
```

```
where Student_Id = 1
```

```
order by term desc
```

```
select @name as name, @score as name
```

حال مقادیر قرار گرفته در متغیرهای @name و @score را با دو روش مقدار دهی select و set بررسی می کنیم. با این پیش فرض که سریالی را برای دانشجو فیلتر کرده ایم که اصلاً وجود ندارد.

```
declare @name varchar ( 100 ) = "
```

```
declare @score real = 18.0
```



```
select top 1 @name = firstname, @score = score
```

```

from sct
    inner join students s on sct.student_id = s.id
where sct.student_id = 10
order by term desc

select @name as name, @score as score

go

declare @name varchar ( 100 ) = ''
declare @score real = 18.0

set @name = (
    select top 1 firstname
    from sct join students s on sct.student_id = s.id
    where Student_Id = 10
    order by term desc
)

set @score = (
    select top 1 score
    from sct join students s on sct.student_id = s.id
    where Student_Id = 10
    order by term desc
)

select @name as name, @score as score

```

نتیجه select اول:

<u>Name</u>	<u>score</u>
	18

نتیجه select دوم:

<u>Name</u>	<u>score</u>
Null	null

Function (۱۶)

توابع تعریف شده توسط کاربر توابعی هستند که تعدادی پارامتر را دریافت نموده و پس از انجام یک عملیات (محاسبه یک مقدار یا جداسازی بخشی از رکوردها) نتیجه را بر می گردانند.

نکته : در سیستم توابع دیگری وجود دارد که قبلاً هم به آنها اشاره شد که توابع سیستمی هستند، در این قسمت به توابعی می پردازیم که کاربر تعریف می کند.



مزایای استفاده از توابع:

امکان برنامه نویسی ماژولار را فراهم می آورند، به این معنی که تابع را یکبار می سازید و در پایگاه داده ذخیره می کنید اما می توانید از آن به دفعات استفاده کنید.

اجرای برنامه سریعتر می شود، چون تابع یکبار کامپایل و در پایگاه داده ذخیره می شود و در اجراهای بعدی نیازی به کامپایل مجدد کد تابع نیست. البته در شرایطی که به درستی از تابع استفاده شود این مزیت برقرار است در غیر اینصورت در زمان استفاده بیجا یا بیش از حد توابع، حتی می توانند باعث کندی شوند.

Table-Valued Function (۱۶-۱)

خروجی این توابع به شکل جدول است و از این جهت بسیار شبیه به view هستند با این تفاوت که یکسری پارامتر به عنوان ورودی دریافت می کنند.

توابع Tabled-valued به دو نوع inline و multi-statement تقسیم می شوند. در نوع multi-statement یک متغیر جدولی تعریف می کنند و از آن برای ایجاد خروجی استفاده می شود، اما در inline متغیر جدولی تعریف نمی شود.

:Inline

```
CREATE FUNCTION test ( @date char ( 10 ) ) RETURNS TABLE
AS
RETURN ( SELECT ... )
```



:Multi-statement

```
CREATE FUNCTION test ( @date char ( 10 ) ) RETURNS @table TABLE ( i int, name
varchar ( 50 ) ) AS
BEGIN
    Body
RETURN
END
```



به عنوان مثال تابعی که سریال دانشجو را بگیرد و لیست درسهای که قبول نشده را به عنوان خروجی نمایش دهد.

```
CREATE FUNCTION dbo.fn_studentCourseStaus ( @student_id int )
RETURNS TABLE
AS
RETURN
```



```
    select c.Code, c.name, sct.score
    from sct
        inner join Courses c on sct.Course_Id = c.Id
    where sct.Student_Id = @student_id and score < 10
go
```

```
select * from dbo.fn_studentCourseStaus ( 1 )
```

توابع ساخته شده را می توانید در

Object explorer\databases\university\programmability\functions\table-valued functions

ببینید. همچنین با راست کلیک کردن می توانید function جدید تعریف کنید.

مثال : تابعی بنویسید که تعداد کل درسهای یک دانشجو اخذ کرده است، تعداد درسهای که پاس کرده و تعداد درسهای که پاس نکرده را نمایش دهد.

```
CREATE FUNCTION dbo.fn_studentCourseStaus2 ( @student_id int )
RETURNS TABLE
AS
RETURN
```



```
    with cte as
    (
        select count (*) cnt, (
            select count (*)
            from sct
                inner join Courses c on sct.Course_Id = c.Id
            where sct.Student_Id = @student_id and
score < 10
        ) nok
        from sct
            inner join Courses c on sct.Course_Id = c.Id
        where sct.Student_Id = @student_id
    )
    select cnt, cnt-nok ok, nok
    from cte
go
select *
from dbo.fn_studentCourseStaus2 ( 1 )
```

Scalar-Valued Function (۱۶-۲)

خروجی این توابع بصورت جدول نیست و فقط یک مقدار که نوع آن در مقابل returns مشخص شده را برمی گردانند.

```
CREATE FUNCTION function_name(parameters list)
RETURNS int [دیگر های داده نوع یا]
AS
begin
    declare @x int
    (expression)

    RETURN @x
end
go
```



نکته: با توجه به اینکه خروجی این فانکشن یک مقدار است ، می توان آنرا در قسمت Select نیز استفاده کرد.



تمرین



- 34- تابعی بنویسید که کد دانشجو را بگیرد و بگوید چند ترم مشغول به تحصیل است.
- 35- تابعی بنویسید که ترم را به عنوان ورودی بگیرد و پرداخت استفاده نشده و بدهی تسویه نشده و بدهی تسویه شده دانشجویان در آن ترم را برگرداند. (سوال * دار)
- 36- لیست دانشجویان در سالهایی که بیش از ۳ درس اخذ کرده اند به همراه مبلغ تسویه نشده و پرداختی استفاده نشده در همان سال را بدست آورید (برای قسمت مالی فانکشن نوشته سپس از دستور apply استفاده کنید) (سوال * دار)

دستورات Rank در SQL Server (۱۷)

توابع Ranking باعث می شود تا به هر یک از سطرهاى جدول یک مقدار به عنوان امتیاز اختصاص داده شود. این مقدار ممکن است منحصر به هر سطر باشد یا این که چند سطر می تواند دارای مقدار مشابه باشد، این بستگی به تابع مورد استفاده داشت.

توجه داشته باشید که مقدار مربوط به این ستون در خروجی Query ها ممکن است در هر بار اجرا متفاوت باشد (البته در صورتی که اطلاعات پایگاه داده شما تغییر یابد یا محاسباتی بر اساس تاریخ و زمان داشته باشید).

ساده ترین مثال، افزودن شماره ردیف به خروجی Query می باشد که توسط تابع ROW_NUMBER() انجام می شود و هر سطر یک مقدار منحصر به فرد خواهد داشت.

در این مقاله در مورد چهار تابع امتیازدهی یا Ranking موجود در SQL Server صحبت خواهیم کرد. همچنین مثال هایی را برای آنها خواهیم نوشت و در مورد تفاوت های آن ها نیز شرح داده خواهد شد که امیدوارم مفید واقع شود.

۱-۱۷) نحوه عملکرد دستور Ranking در SQL Server

این توابع مقادیر خود را بر اساس یک مرتب سازی تولید می کنند. به مثال زیر توجه نمایید:

```
SELECT ID, ROW_NUMBER () OVER (PARTITION BY ID ORDER BY Grade ) AS  
RowNumber  
FROM SampleTable
```

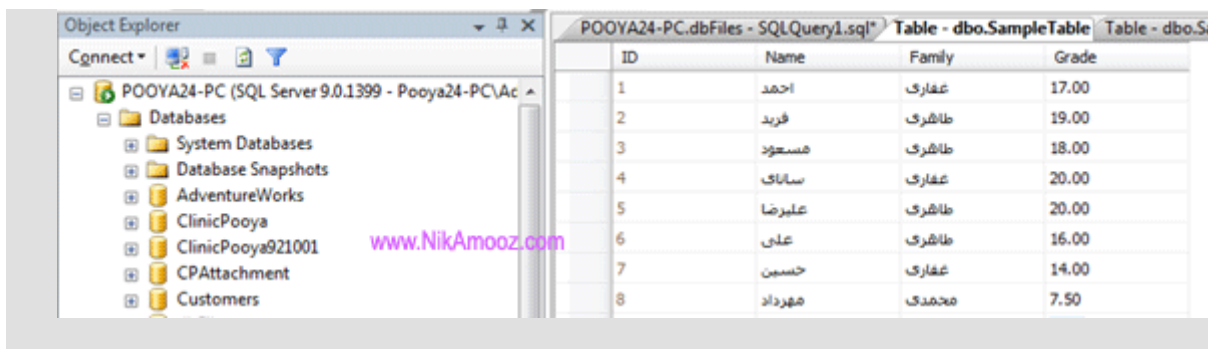


عبارت PORTITION BY که بعد از ROW_NUMBER () OVER نوشته می شود برای دسته بندی بر اساس فیلد مورد نظر استفاده می شود. به عبارت دیگر، عملگر PARTITION داده ها را به دسته های مختلفی شکسته و عمل مورد نظر را برای هر دسته یا هر پارتیشن به صورت اختصاصی انجام می دهد. برای مثال وقتی از تابع Row_Number به صورت تنها استفاده می شود، یک ترتیب شماره گذاری از یک تا N (تعداد رکوردها) به هر رکورد داده می شود. اما وقتی از دستور Row_Number به همراه PARTITION استفاده می شود، در هر پارتیشن به صورت جداگانه از یک تا N (تعداد سطرهاى پارتیشن) به هر سطر داده می شود.

همان گونه که مشاهده می کنید بعد از PORTITION BY عبارت ORDER BY ID نوشته شده است که مشخص می کند این شماره ردیف بر اساس چه ترتیبی به سطرها اختصاص داده شود.

در این مثال زمانی از عبارت PORTITION BY ID استفاده می کنیم که بخواهیم بر اساس کمترین نمره هر دانشجو شماره ردیف بگیرد.

برای مثال های این مقاله از جدول شکل زیر استفاده خواهیم نمود.



ID	Name	Family	Grade
1	احمد	عفاری	17.00
2	فرید	طاهری	19.00
3	مسعود	طاهری	18.00
4	سانای	عفاری	20.00
5	علیرضا	طاهری	20.00
6	علی	طاهری	16.00
7	حسین	عفاری	14.00
8	مهرداد	محمدی	7.50

۲-۱۷) تابع ROW_NUMBER()

اگر بخواهیم به خروجی یک دستور SELECT شماره سطر را نیز اضافه نماییم در این حالت بایستی از این تابع استفاده نماییم.

مثال:

```
SELECT ID, Name, Family,
ROW_NUMBER () OVER (ORDER BY ID) AS RowNumber
FROM SampleTable
```



خروجی:

	ID	Name	Family	RowNumber
1	1	سیده	فروزان	1
2	2	فاطمه	ترابی	2
3	3	ندا	اکبری	3
4	4	سعیده	اسدی	4
5	5	عاطفه	صادقی	5
6	6	علی	قاسمی	6
7	7	علی	هادی پور	7
8	8	مرجان	تقریبیان	8
9	9	زهرا	جعفرآبادی	9
10	10	فرناز	فرضی	10
11	11	وحید	قاسم پورمقانی	11

همان طور که می بینید ستون RowNumber به خروجی ما اضافه شده است. ممکن است بگویید مقدار این ستون که برابر با ستون ID است، چرا از این تابع استفاده کردیم؟ بله حق با شماست، در این مثال ما می توانستیم از فیلد ID استفاده کنیم. منتها چون این فیلد به صورت IDENTITY تعریف شده است بنابراین احتمال زیادی دارد که مقادیر دقیقا به ترتیب نباشند. در این حالت این فیلد کارایی نخواهد داشت.

۳-۱۷) دستور RANK در SQL Server

این تابع امتیازی از ۱ تا الی آخر به هر سطر از خروجی دستور SELECT ما می دهد. اگر دو سطر دارای مقادیر برابر باشند در این صورت مقدار تابع Rank برای هر دو سطر برابر خواهد بود. به مثال زیر دقت کنید :

اگر اطلاعات جدول ما به این شکل باشد:

ID	Name	Family	Grade
1	احمد	عفاری	17.00
2	فرید	طاهری	19.00
3	مسعود	طاهری	18.00
4	سانای	عفاری	20.00
5	علیرضا	طاهری	20.00
6	علی	طاهری	16.00
7	حسین	عفاری	14.00
8	مهرداد	محمدی	14.00
9	NULL	NULL	NULL

بعد از اجرای دستور زیر:

```
SELECT ID, Name, Family, Grade,
RANK() OVER (ORDER BY Grade) AS Rank
FROM SampleTable
```



	ID	Name	Family	Grade	Rank
1	4	سعیده	اسدی	12	1
2	8	مرجان	تقریبیان	13	2
3	11	وحید	قاسم پورمقانی	17	3
4	2	فاطمه	ترابی	18	4
5	5	عاطفه	صادقی	19	5
6	6	علی	قاسمی	19	5
7	7	علی	هادی پور	20	7
8	3	ندا	اکبری	20	7
9	9	زهرا	جعفرآبادی	20	7
10	10	فرناز	فرضی	20	7
11	1	سپیده	فروزان	20	7

خروجی زیر حاصل خواهد شد:

بر اساس فیلد Grade مقداری به عنوان Rank اختصاص داده شده است. همچنین برای مقادیر یکسان Rank های برابر تخصیص داده شده است. به سطر هفتم در خروجی دستور دقت کنید، مقدار ستون Rank برابر با ۷ می باشد در حالی که بعد از Rank های با مقدار ۵ باید مقدار ۶ را به ما می داد. این خاصیت تابع Rank() می باشد.

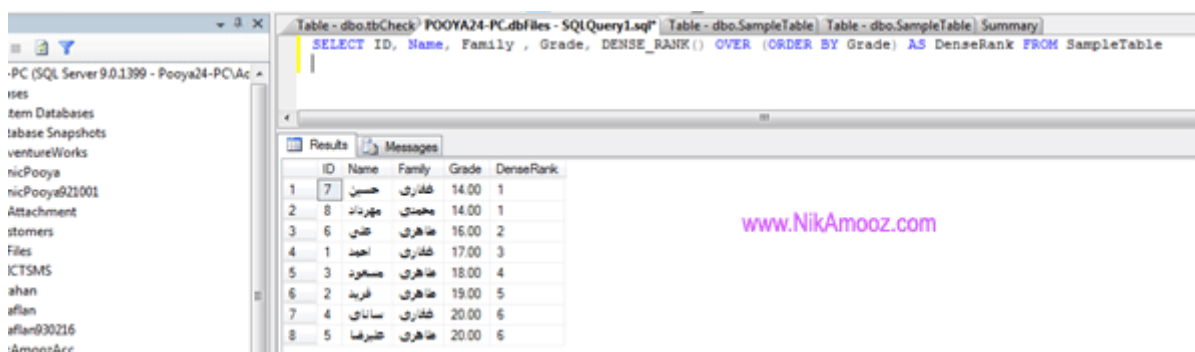
۴-۱۷) دستور DENSE_RANK()

عملکرد این تابع دقیقاً همانند تابع RANK() می باشد، با این تفاوت که مقادیر مربوط به ستون DENSE_RANK() به صورت مرتب به سطرها اختصاص داده می شود. به مثال زیر دقت کنید:

```
SELECT ID, Name, Family, Grade, DENSE_RANK () OVER (ORDER BY Grade)
AS DenseRank
FROM SampleTable
```



با اجرای این دستور روی اطلاعات جدول بالا خروجی زیر حاصل خواهد شد:



ID	Name	Family	Grade	DenseRank
1	7	حسن	14.00	1
2	8	مهرمان	14.00	1
3	6	علی	16.00	2
4	1	احمد	17.00	3
5	3	مسعود	18.00	4
6	2	فرید	19.00	5
7	4	سنا	20.00	6
8	5	علیرضا	20.00	6

اگر به سطر سوم در خروجی دستور دقت کنید، مقدار ستون DenseRank برابر با ۲ می باشد و بقیه مقادیر به صورت متوالی به این ستون اختصاص داده شده است. تفاوت دو تابع Rank و Dense_Rank همین مورد می باشد.

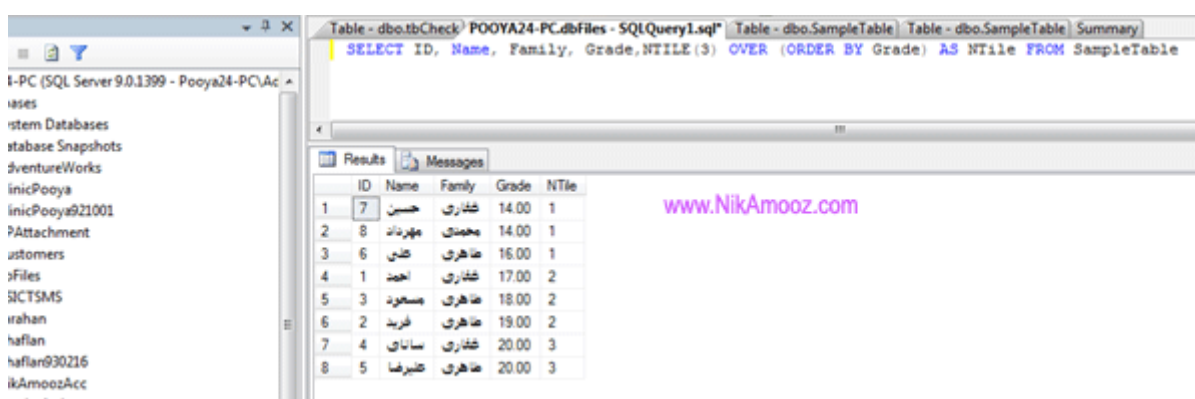
۵-۱۷) تابع NTILE(n)

این تابع برای دسته بندی کردن اطلاعات استفاده می شود. تعداد دسته ها به صورت یک پارامتر به این تابع داده می شود. به مثال زیر دقت کنید:

```
SELECT ID, Name, Family, Grade,
NTILE ( 3 ) OVER ( ORDER BY Grade ) AS NTile
FROM SampleTable
```



خروجی حاصل از اجرای این دستور به شکل زیر خواهد بود:



ID	Name	Family	Grade	NTile
1	7	حسن	14.00	1
2	8	مهرمان	14.00	1
3	6	علی	16.00	1
4	1	احمد	17.00	2
5	3	مسعود	18.00	2
6	2	فرید	19.00	2
7	4	سنا	20.00	3
8	5	علیرضا	20.00	3

در مثال فوق، پارامتر این تابع برابر ۳ می باشد، یعنی اطلاعات را به سه دسته تقسیم می کند. توجه داشته باشید که در این تقسیم بندی دسته های با تعداد سطر بزرگتر در ابتدای خروجی نمایش داده می شود.

تمرین



- 37- مجموع معدل ۳ دانشجوی برتر را اعلام کنید. (برتر: دارای بالاترین معدل در یک ترم) به ازاء هر ترم. (سوال * دار)
- 38- دو دانشجوی برتر را به ازای هر ترم نمایش دهید. (از دو روش rank و apply انجام دهید و plan بگیرید) (سوال * دار)
- 39- سعی کنید با کمک گرفتن از اینترنت راهی را پیدا کنید که با استفاده از Recursive CTE بتوان اعداد ۱ تا ۲۵۶ را تولید کرد)

Stored Procedures (۱۸)

رویه های ذخیره شده یا stored procedures یکسری ورودی دریافت می کند و عملیاتی که برایش تعریف شده را انجام می دهد و خروجی را در صورت وجود بشکل جدول و یا پارامتر Out بر می گرداند. می توان در آن از دستورات مختلف موجود در SQL استفاده کرد.

```
Create procedure procedure_name (parameters list)
```

```
as
```

```
begin
```

```
    Expressions
```

```
end
```

```
Go
```



نکته: برای مشخص کردن بدنه پروسیجر می توان از Begin و End در ابتدا و انتهای دستورات بعد از as استفاده کرد. اما توجه داشته باشید که end نشانه ی پایان پروسیجر نیست بلکه با دستور Go انتهای یک پروسیجر را می توان مشخص کرد.



برای اجرا شدن یک SP از دستور زیر استفاده می کنیم:

```
exec procedure_name parameters value
```



به عنوان مثال برای ثبت نام دانشجوی جدید می توان sp زیر را نوشت:

```
Create procedure sp_insertStudents
```

```
(
```

```
    @firstname varchar(50),
```

```
    @lastname varchar(100),
```

```
    @code varchar(20),
```

```
    @address varchar(1000),
```

```
    @tel varchar(20),
```

```
    @reshte varchar(100)
```

```
)
```

```
as
```

```
Insert into students ( FirstName, LastName, Address, Tel, Code, Reshte_Id )
```

```
SELECT top 1 @FirstName, @LastName, @Address, @Tel, @Code, id
```

```
FROM reshte
```

```
where name = @reshte
```



```
exec sp_insertStudents 'zahra', 'mohammadi', '3005', 'sanat', '7777777', 'computer'
```

لیست دانشجویانی را نمایش دهید که در ترم مورد نظر درس مشخصی را اخذ نکرده اند. (ترم و درس به عنوان ورودی داده می شوند)

```
alter procedure sp_StudentsTermCourse ( @term int, @courseId int )
```

```
as
```

```
select distinct s.*
```

```
from Students s
```

```
inner join sct on s.id = sct.student_id
```

```
where term = @term and not exists (
```

```
select 1
```

```
from sct
```

```
where student_id=s.id and course_id =  
@courseId and term =@term
```

```
)
```

```
go
```

```
exec sp_StudentsTermCourse 1, 10
```

برای استفاده از پارامتر out باید در مقابل پارامتری که می خواهید خروجی بر گرداند عبارت out را نوشته و در پروسیجر مقدار آنرا مشخص کنید، و در نهایت برای استفاده از مقدار آن باید به شکل زیر عمل کنید:

```
create procedure testOutParameter ( @outParameter varchar ( 100 ) out, @text
```

```
varchar ( 100 ) )
```

```
as
```

```
select @outParameter = isnull ( @text, 'Hi' )
```

```
go
```

```
declare @result varchar ( 100 )
```

```
exec testOutParameter @result out, null
```

```
select @result as outParam
```

نتیجه

outParam

Hi

کار با if (۱۹)

برای بررسی یک یا چند شرط و انجام عملیات مختلف به ازاء هر شرط از دستور if می توان استفاده کرد. این دستور بر خلاف case می تواند به تنهایی استفاده شود.

```
شرایط if
begin
    دستورات
end
else
begin
    دستورات
end
```



در صورت درست بودن شرایط مقابل دستور if عملیاتی که بعد از آن قرار گرفته است اجرا خواهند شد و در غیر اینصورت عملیات تعریف شده در قسمت else اجرا خواهد شد. البته قسمت else اجباری نیست و می تواند نباشد.

نکته: اگر بعد از if و یا else فقط یک دستور باشد نیازی به گذاشتن begin و end نیست.



```
declare @x int, @y varchar ( 20 )

if @x = 1
    set @y = 'است قرار بر شرط'
else
    set @y = 'است نشده برقرار شرط'
select @y
```



در صورتی که متغیر X با ۱ مقدار دهی شود شرط if درست خواهد بود و عبارت "شرط بر قرار است" نمایش داده خواهد شد و اگر متغیر X به هر عددی غیر از ۱ ست شود وارد قسمت Else شده و عبارت "شرط برقرار نشده است" نمایش داده خواهد شد.

مثال:

```
alter procedure sp_StudentsTermCourse ( @term int, @courseId int, @pass bit )
as
    if ( @pass = 1 )
        select distinct s.*
        from Students s
        inner join sct on s.id = sct.student_id
        where term = @term and course_id = @courseId

    else
        select distinct s.*
        from Students s
        inner join sct on s.id = sct.student_id
        where term = @term and not exists (
            select 1
            from sct
            where student_id = s.id and
            course_id = @courseId and
            term=@term
        )

go
```



`exec sp_StudentsTermCourse 1, 10, 0`

پروسیجر قبلی را به شکلی تغییر دادیم بصورتی که پارامتر جدید @pass اگر صفر وارد شود لیست دانشجویانی را می آورد که در ترم تعیین شده، درس مشخص شده را نگذرانده باشند و اگر یک وارد شود لیست دانشجویانی که درس را در آن ترم اخذ کرده اند را بر می گرداند.

کار با While (۲۰)

برای حلقه در SQL می توان از دستور while استفاده کرد. تا وقتی که شرط برقرار است، اجرای دستورات تعریف شده در حلقه تکرار می شود.

WHILE Boolean_expression

{ sql_statement | statement_block | BREAK | CONTINUE }



مثال:

alter procedure sp_testWhile

as

```
select row_number () over ( order by term ) as rowNO, avg ( score ) GPA,
term, student_id
into #temp
from sct
group by term, student_id
```

```
declare @cnt int = 0
```

```
set @cnt = ( select count ( * ) from #temp )
```

```
WHILE @cnt > 0
```

```
BEGIN
```

```
print @cnt
if ( select gpa
      from #temp
      where rowno=@cnt
      ) between 15 and 17
update sct
set score = score + 1
where student_id = ( select Student_Id
                    from #temp
                    where rowno = @cnt
                    )
else if ( select gpa
          from #temp
          where rowno = @cnt
          ) > 17
```

```
select s.*, gpa, term
from students s
```



```

inner join #temp on id = Student_Id
where rowno = @cnt
--else
--break or continue
set @cnt = @cnt - 1

END

go

```

drop table #temp

پروسیجر بالا معدل هر دانشجو را در هر ترم محاسبه می کند و در حلقه ای به ازاء تک تک دانشجویان بررسی می کند اگر معدل آنها بین ۱۵ تا ۱۷ باشد به هر درس آن دانشجو یک نمره اضافه می کند و در غیر اینصورت اگر معدل بزرگتر از ۱۷ باشد اطلاعات دانشجو را نمایش می دهد.

تمرین



40- پروسیجر بنویسید که دانشجو، ترم، نوع پرداخت و مبلغ را به عنوان ورودی بگیرد. بررسی کند اگر دانشجو در آن ترم درس اخذ کرده بود یک ردیف به جدول پرداخت اضافه کند با شرح تخفیف و اگر دانشجو در آن ترم ثبت نام نکرده بود مبلغ را بین پرداخت های قبلی او به طور مساوی تقسیم کند.

41- پروسیجر بنویسید که بررسی کند در ترم X دانشجوی ممتاز کیست و ۱۵٪ کل مبلغ آن ترم را محاسبه کند و یک فیش نقدی برای آن مبلغ در آن ترم برای دانشجوی ممتاز ثبت کند. (سوال * دار)

42- پروسیجر بنویسید که سال را به عنوان ورودی گرفته و در جدول ترم، اطلاعات سال قبل رابدست آورده مبالغ را به اضافه ۳۰٪ کنید و برای سال جدید ذخیره کنید.

43- پروسیجر بنویسید که لیست دانشجویان بستانکار را به همراه مبلغ بستانکاری آنها را به شکل زیر نمایش دهد.

دانشجو با کد دانشجویی محصل در رشته ی مبلغ بستانکار است.

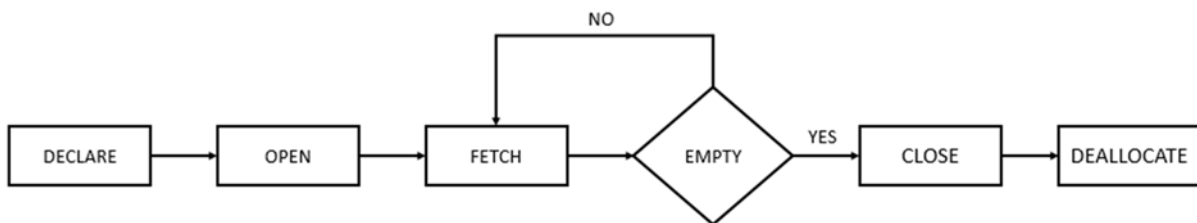
Cursor (۲۱)

نکته: تا جایی که ممکن است از cursor استفاده نکنید. چون از لحاظ کارایی بسیار پایین است و موجب کندی شدید در query میگردد.



یک متغیر cursor در واقع یک ارجاع به تعریف cursor می باشد. هنگام استفاده از دستورات SQL نظیر Select کلیه رکوردهای درخواستی بطور کامل استخراج می گردد اما در مواردی نیاز است که رکوردهای استخراج شده تحت شرایطی خاص مورد پردازش مجدد قرار گرفته و به برنامه درخواست کننده ارسال گردد در این صورت استفاده از کرسرها بسیار حائز اهمیت خواهد بود. برای استفاده از یک کرسر می توان به ترتیب مراحل ذیل عمل نمود.

- یک متغیر از نوع کرسر تعریف می گردد. که شامل دستور Select درخواستی خواهد بود.
- با استفاده از دستور Open یک کرسر آماده استفاده می گردد.
- با استفاده از دستور Fetch حرکت درون یک کرسر امکان پذیر می گردد که در این حالت مقدار فیلدهای اعلام شده در رکورد جاری در دسترس می باشد.
- با استفاده از دستور Close کرسر فعال شده بسته می شود.
- با استفاده از Deallocate فضای اختصاص داده شده برای کرسر آزاد می گردد.



```
DECLARE @cursor CURSOR
SET @cursor = CURSOR FOR SELECT * FROM tableName
OPEN @cursor
FETCH @cursor
WHILE ( @@FETCH_STATUS = 0 ) BEGIN
    body
    FETCH @cursor
END
CLOSE @cursor
DEALLOCATE @cursor
```



نکته: سعی کنید تا جایی که امکان دارد از کرسر استفاده نکنید، چرا که باعث کاهش کارایی خواهد شد. یکی از دلایل اینکه از cursorها صرف نظر می شود این است که در حلقه ی کرسر هر عبارت fetch معادل با یک عبارت select است که هزینه ی زیادی خواهد داشت.



```
declare @name varchar ( 100 ), @family varchar ( 100 )
```

```
DECLARE @cursor CURSOR
SET @cursor = CURSOR FOR SELECT FirstName, LastName
FROM students
```



```
OPEN @cursor
FETCH @cursor into @name,@family
WHILE ( @@FETCH_STATUS = 0 ) BEGIN
```

```
    select ' دانشجو اسم ' + @family + ' ' + @name + ' است '
```

```
FETCH @cursor into @name, @family
END
CLOSE @cursor
DEALLOCATE @cursor
```

در واقع هر رکورد را بصورت جداگانه در قسمت while داریم و می توانیم عملیاتی را که نیاز داریم با توجه به اطلاعات تک تک فیلد هایش انجام دهیم.

تمرین



44- کرسری بنویسید که در جدول پرداخت ها به ازاء پرداخت هایی که توضیحات ندارند، در فیلد توضیحات آنها بنویسد چند درس عملی و چند درس تئوری در آن ترم اخذ کرده است. (سوال * دار)

Trigger (۲۲)

یک تریگر نوع خاصی از sp است که هنگامی که اتفاق خاصی در اس کیوال سرور می افتد بطور اتوماتیک اجرا میشود. برای مثال میخواهیم زمانی که یک رکورد به جدول A اضافه شد، به صورت اتوماتیک اتفاق خاص دیگر (مثل درج یک رکورد در جدول B) به صورت اتوماتیک رخ دهد. فرمت کلی تریگر به صورت زیر است:

```
CREATE TRIGGER [ schema_name . ] trigger_name
ON { table | view }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS
```



۱- (۲۲-۱) FOR, AFTER, INSTEAD OF

باید در تعریف Trigger مشخص شود که عمل تعریف شده در چه زمانی انجام شود. در مثال قبلی میتوان تعریف کرد که «بعد از» درج یک رکورد در جدول A یک رکورد در جدول B درج نماید، یا در حالتی دیگر میتوان تعریف کرد که «به جای» درج یک رکورد در جدول A یک رکورد در جدول B درج نماید. این که این عمل «بعد از» و یا «به جای» عمل خاصی صورت گیرد با استفاده از For, AFTER, INSTEAD OF انجام میشود.

- For: «بعد از» درج در A در B را انجام میدهد.
- After: دقیقاً مانند for عمل میکند.
- Instead Of: «به جای» درج در A در B را انجام میدهد. به جای اجرای دستورات تعریف شده بر روی جدول یا view مورد نظر، تریگر فعال می گردد. البته ممکن است تغییرات بعداً از طریق دستورهای تریگر در جدول یا view مورد نظر اعمال شود.

در View تنها می توان از Instead Of استفاده کرد.



```
CREATE TRIGGER RT
ON tablename
INSTEAD OF DELETE
AS
PRINT 'CAN NOT DELETE '
```



۲- (۲۲-۲) انواع تریگر:

- INSERT trigger: بعد از اضافه کردن یک رکورد در جدول مورد نظر این تریگر اجرا می شود.
- update trigger: بعد از آپدیت کردن یک رکورد در جدول مورد نظر این تریگر اجرا می شود.
- delete trigger: بعد از حذف یک رکورد در جدول مورد نظر این تریگر اجرا می شود.

بدنه تریگر:

پس از کلمه AS بدنه تریگر قرار می گیرد که در آن عملی که تریگر قرار است انجام دهد ، نوشته می شود.

۲۲-۳ Deleted و Inserted جداول

در هنگام کار با Triggerها، SQL به صورت پیش فرض، جداولی داخلی به نامهای Deleted و Inserted دارد که در طول انجام یک عمل از آنها استفاده میکند. برای مثال زمانی که یک رکورد در جدول A درج میکنیم، SQL به صورت خودکار این رکورد را در جدول Inserted قرار میدهد. پس از اجرا شدن Trigger روی جدول B و درج شدن یک رکورد؛ آنگاه جدول Inserted هم خالی میشود. در واقع این دو جدول، جداول موقتی هستند که درج در آنها در اختیار SQL Server است. به همین ترتیب زمانی که یک رکورد را Delete می‌کنیم، تا زمان اجرا شدن تمام Triggerهای مرتبط با آن، رکورد حذف شده در جدول Deleted نگه‌داری خواهد شد. همچنین یک Update شامل یک حذف و یک درج جدید خواهد بود که حاصل آن درج در هر دو جدول Inserted و Deleted خواهد بود. لذا زمانی که درج در جدول B با استفاده از اطلاعات موجود در رکورد اضافه شده در جدول A باشد، میتوان از این جداول میانی برای واکنشی داده‌های مورد نظر استفاده کرد.

مثال:

```
Create TRIGGER tr_InsertDepts
ON EDU.Selection
AFTER Insert
AS
    insert into ml.Debt
    select ins.StudentId, ins.Id, u.TermId, u.Price
    From inserted ins
    join edu.Unit u on u.Id = ins.UnitId
```



۲۲-۴ استفاده از تابع update(column):

مقدار این عبارت true یا false بوده و نشان می‌دهد 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 می‌باشد. به کمک این عبارت و دستور if می‌توانیم در مقابل تغییر بعضی از فیلدهای مورد نظر عکس العمل مناسب پیش بپیماییم) در واقع دستور UPDATE (column) چک می‌کند که آیا ستون مورد نظر تغییر کرده است یا خیر

مثال:

```
Create TRIGGER tr_UpdateDepts
ON EDU.Selection
AFTER Update
AS
    if ( update ( unitid ) )
    BEGIN
        Update ml.Debt
        set Value = value + NewUnit.Price - lastUnit.Price
        From inserted ins
        inner join deleted del on ins.Id = del.Id
        inner join edu.Unit u on u.Id = ins.UnitId
        inner join Unit lastUnit on del.UnitId = lastUnit.Id
        inner join Unit NewUnit on ins.UnitId = NewUnit.Id
    END
```



دربنده تریگرنمی توان از دستورات CREATE DATABASE ، ALTER DATABASE و DROP DATABASE استفاده کرد .



در تریگر در صورت مشخص نکردن delete، update، insert را پیش فرض قرار می دهد



۵-۲۲ حذف و تغییر تریگرها:

برای تغییر و یا حذف یک تریگر به ترتیب از دستورات ALTER TRIGGER و DROP TRIGGER استفاده می شود. برای فعال یا غیرفعال کردن یک تریگر از دستورات ENABLE و DISABLE استفاده می شود.

`<ENABLE|DISABLE> TRIGGER <ALL|trigger_name>`



تمرین



45- یک ستون به اسم Remain به جدول بدهی اضافه و مقادیر داخل آن را بروزرسانی نمایید. یک تریگر بنویسید که در صورتی که یک تسویه برای یک بدهی ثبت شد، آنگاه باقی بدهی تسویه نشده را در فیلد Remain به روز رسانی کند. (برای حذف و بروزرسانی تسویه هم تغییرات در فیلد Remain اعمال شود)

Transaction (۲۲)

یک تراکنش مجموعه ای از دستورات است که تغییری را در پایگاه داده ایجاد می کند. نکته مهم در مورد تراکنش این است که مجموعه دستورات موجود در آن یا بصورت کامل ایجاد می شود یا هیچ کدام از آنها به اجرا در نمی آید تا در نهایت سازگاری داده ها محفوظ بماند.

فرض کنید در یک دستگاه خودپرداز، دستور انتقال مبلغی را از حساب خود به حساب دیگری صادر کرده اید. اگر عملیات اول که کسر مبلغ از حساب شماست اجرا شود اما بنا به دلایلی عملیات واریز به حساب مقصد صورت نگیرد، در عمل چه اتفاقی می افتد؟ پاسخ روشن است، پایگاه داده در وضعیت ناسازگار قرار می گیرد و صحت داده های موجود در آن از بین می رود.

برای غلبه بر چنین مشکلی، مجموعه دستورات را درون یک تراکنش قرار می دهند تا در صورت اجرای موفق همه ی دستورات، عمل تثبیت (Commit) صورت گیرد و در غیر این صورت همه تغییرات برگردانده (Rollback) شوند. الگوی یک تراکنش به شکل زیر است:

نام تراکنش Begin Tran

مبلغ را از موجودی حساب اول کم کن

مبلغ را به موجودی حساب دوم اضافه کن

نام تراکنش Commit Tran در صورت عدم وقوع خطا

نام تراکنش Rollback Tran در صورت بروز خطا

مثال:

```
Create Table Table1 ( No int )
```

```
BEGIN TRAN T1
```

```
INSERT INTO Table1 VALUES ( 1 )
```

```
INSERT INTO Table1 VALUES ( 2 )
```

```
ROLLBACK TRAN T1
```

```
INSERT INTO Table1 VALUES ( 3 )
```

```
SELECT * FROM Table1
```



دستورات بالا را یکبار با ROLLBACK TRAN T1 اجرا کنید و یکبار بدون آن، خواهید دید که دستور rollback تغییرات را بر می گرداند. برای درک بهتر این مسئله قبل از rollback هم یک select از جدول table1 بگیرید.

مفهوم ACID :

تجزیه ناپذیر بودن (Atomicity)، سازگاری (Consistency)، جداسازی (Isolation) و پایداری (Durability)

Atomicity: تمام تغییرات داده در داخل تراکنش می بایست با موفقیت انجام شود یا اینکه هیچ یک از تغییرات انجام نشود (همه یا هیچ کدام). به عبارتی عملیات دستکاری داده ها در تراکنش یک واحد به حساب می آیند و غیر قابل تجزیه هستند. یا همه با هم با موفقیت اجرا می‌شوند یا اینکه هیچ کدام اجرا نمی شوند.

Consistency: تراکنش ها، سازگاری و جامعیت داده های پایگاه داده (database) را حفظ می کنند به بیان دیگر، تراکنش، پایگاه داده را از یک حالت سازگار (consistent) به حالت سازگار دیگری تبدیل می کند. به این معنا که در

صورت roll back شدن تراکنش (با شکست مواجه شدن یکی از دستورات) پایگاه داده به حالت (state) سازگاری قبل از اجرا شدن تراکنش بر میگردد. یا بعد از پذیرفته شدن تراکنش (commit) یک وضعیت سازگاری جدید شکل میگیرد. در نتیجه هیچ گاه پایگاه داده را در حالت ناسازگار رها نخواهد کرد.

Isolation : تغییرات هر تراکنش مستقل از سایر تراکنش ها می باشد .

تراکنش ها جدا از یکدیگر هستند به بیان دیگر، هر چند به طور کلی ممکن است چند تراکنش به طور همزمان اجرا شوند اما به هنگام سازی هر یک از این تراکنش ها از بقیه ی تراکنش ها پنهان نگه داشته می شود تا وقتی که تراکنش پذیرفته شود. به عبارت دیگر برای دو تراکنش متمایز T1 و T2 ، تراکنش T1 می تواند به هنگام سازی های T2 را ببیند (پس از پذیرفته شدن T2) یا تراکنش T2 می تواند به هنگام سازی های T1 را ببیند (پس از پذیرفته شدن T1) اما هر دو به طور همزمان نمی توانند به هنگام سازی های یکدیگر را ببینند .

Durability : پس از آن که تراکنشی پذیرفته شد (committed) به هنگام سازی های آن، در پایگاه داده باقی می ماند و state جدید پایگاه داده قابل دسترسی بوده، حتی اگر سیستم اندکی بعد به دلیل مشکلات سخت افزاری و یا نرم افزاری از کار بیفتد.

در SQL Server عملاً این ترمیم و بازسازی توسط checkpoint ها صورت گرفته و رویه ترمیم پایگاه داده (recovery) در هنگام startup انجام میشود.

نکته: انواع تراکنش:

خودکار (automatic) یا (autocommit) ، تراکنش ها بصورت پیش فرض از این نوع هستند، هر تراکنش در انتها یا commit می شود و یا rollback، در صورت موفقیت اجرای دستورات تراکنش commit شده در غیر اینصورت rollback خواهد شد. ضمنی (implicit)، تعرف شده توسط کاربر (user_defined)، توزیع شده (distributed).



نکته: امکان تعریف تراکنش های تو در تو وجود دارد و @@Trancount تعداد تراکنش ها را نشان می دهد.



```
SELECT 'Before BEGIN TRAN', @@TRANCOUNT
BEGIN TRAN
SELECT 'After BEGIN TRAN', @@TRANCOUNT
DELETE students where id = 10
BEGIN TRAN nested
SELECT 'After BEGIN TRAN nested', @@TRANCOUNT
DELETE sct where student_id = 5
COMMIT TRAN nested -- Does nothing except decrement @@TRANCOUNT
SELECT 'After COMMIT TRAN nested', @@TRANCOUNT
GO -- When possible, it's a good idea to place ROLLBACK TRAN in a separate
batch
-- to prevent batch errors from leaving open transactions
ROLLBACK TRAN
SELECT 'After ROLLBACK TRAN', @@TRANCOUNT

SELECT TOP 5 *
FROM sct
```



نتیجه:

Before BEGIN TRAN	0
After BEGIN TRAN	1
After BEGIN TRAN nested	2
After COMMIT TRAN nested	1

نکته: Commit و یا Rollback شدن تراکنش های داخلی وابسته به بیرونی ترین تراکنش است.



نکته: نام تراکنش مقابل دستور rollback نباید به یک تراکنش داخلی از یک مجموعه تراکنش های تو در تو اشاره داشته باشد، بلکه تنها می تواند به بیرونی ترین تراکنش اشاره داشته باشد.



نکته: اگر دستور Rollback بدون نام تراکنش در هر سطحی از یک مجموعه تراکنش های تو در تو اجرا شود، تمام تراکنش های تو در توی مجموعه را rollback می کند.



نکته: Savepoint

در برخی شرایط ممکن است بخواهیم در هنگام ROLLBACK مجدداً به ابتدای تراکنش باز نگردیم تا مجبور باشیم دوباره کار را از ابتدا از سر بگیریم. بعنوان مثال تا قسمتی از تراکنش پیش رفتیم، به خطایی برخورد می کنیم و می خواهیم از نقطه ای خاص از تراکنش کار را از سر بگیریم. در چنین کاربردهایی از SavePoint استفاده می کنیم.



برای روشن تر شدن مفهوم SavePoint فرض کنید قصد داریم بلیطی از تهران به سیدنی رزرو کنیم. برای این منظور ابتدا عمل رزرواسیون را از تهران به دوبی انجام می دهیم و سپس از دوبی به سنگاپور و در نهایت از سنگاپور به سیدنی. حال در این بین می توانیم در نقطه تهران - دوبی SavePoint قرار دهیم تا در صورت بروز هرگونه خطا مجدداً رزرواسیون را از ابتدا آغاز نکنیم. اگر در هنگام رزرو بلیط دوبی - سنگاپور خطایی بروز دهد می توانیم به نقطه تهران - دوبی ROLLBACK کنیم و از آنجا مسیر دیگری را انتخاب کنیم. توجه داشته باشید که ROLLBACK به SavePoint وضعیت پایگاه داده به همان نقطه بازگردانده می شود.

```
begin tran
s1;
SAVE TRANSACTION sp1;
s2;
SAVE TRANSACTION sp2;

if ( condition )
ROLLBACK TRANSACTION sp1;
...
...
commit
```



تمرین

46- پروسیجر هایی که در تمرینات قسمت SP انجام دادید را transactional کنید.



۲۴) به دام انداختن خطا (Error Trapping)

در پروسیجرهای پیچیده اگر خطایی بوجود بیاد چگونه باید آنرا کشف کنیم؟

انواع روش های به دام انداختن خطا:

- @@error
- Try...catch

۱-۲۴) @@error

ابتدایی ترین روش مدیریت خطا بررسی مقدار @@error است. البته این شیوه خیلی کارآمد نیست. یکی از مشکلاتی که در این شیوه مطرح است مربوط به مدت زمان اعتبار مقدار متغیر @@error است. در واقع اعتبار مقدار این متغیر تا قبل از اجرای عبارت بعدی است. لذا اگر در مکانی از کد، احتمال بروز خطا متصور است، حتما از یک متغیر محلی و ثبت مقدار @@error و یا پردازش مستقیم خطای رخ داده استفاده کنید.

مثال:

```
SELECT 1 / 0  
SELECT @@ERROR  
SELECT @@ERROR
```



اولین دستور با شکست مواجه خواهد شد و پیغام خطایی مبنی بر تقسیم بر صفر (Divide by zero) صادر خواهد کرد. پس متغیر @@error حاوی شماره مربوط به خطا خواهد شد (که در اینجا شماره خطا ۸۱۳۴ است).

دومین دستور مقدار متغیر @@error را بر میگرداند (که همان عدد مذکور است) و سپس مقدار صفر به متغیر @@error اختصاص داده می شود (که به معنای اجرای موفقیت آمیز آخرین دستور است).

حالا سومین دستور مقدار ۰ را به جای عدد ۸۱۳۴ برمیگرداند.

این موضوع محدود به اینگونه دستورات نمی شود. بر اساس مستندات Microsoft حتی عبارات شرطی مثل IF مقدار @@error را reset می کنند .

مثال:

```
SELECT 1 / 0  
IF @@ERROR > 0 SELECT @@ERROR
```



اولین @@error آخرین شماره خطا را نگه داشته است که عددی بزرگتر از صفر است. پس شرط دستور IF صحیح است در نتیجه @@error انتخاب می شود، اما دومین @@error که بعد از بررسی دستور IF اجرا می شود بر خلاف تصور شماره خطا مربوط به آخرین دستور که شرط IF باشد را نگهداشته است که عدد صفر است. به بیان دیگر دستور IF مقدار @@error را reset کرده است.

نکته : در این قسمت می توان به XACT_ABORT نیز اشاره کرد، با On و Off کردن آن در هنگام کار با تراکنش ها، در صورت بروز خطا نتیجه کار متفاوت می باشد. برای درک بهتر این موضوع مثالی در زیر مطرح شده است.



دو جدول Tbl1 و Tbl2 را در نظر بگیرید که با هم در ارتباط هستند، و در هنگام وارد کردن ۳ رکورد به جدول Tbl2، یک رکورد را با اطلاعاتی به غیر از آنچه در کلید اصلی جدول Tbl1 می باشد پر می کنیم، که خطا ایجاد کند. در این شرایط اگر XACT_ABORT ، off باشد دو رکورد دیگر در جدول وارد می شوند و رکوردی که به خطا خورده اضافه نخواهد شد. و اگر on باشد، هیچکدام از رکورد ها به جدول اضافه نمی شوند.

```
create table Tbl1 ( id int not null primary key )
create table Tbl2 ( T1Id int references Tbl1 ( id ) )
go
insert into Tbl1 values (1), (3), (5), (6)
go
--*****
set Xact_abort off;
go
begin transaction
insert into Tbl2 values (1)
insert into Tbl2 values (2)
insert into Tbl2 values (3)
commit transaction
go
select *
from Tbl2
```



نتیجه: رکورد های شامل ۱ و ۳ اضافه می شوند اما ۲ باعث بروز خطا خواهد شد و در نتیجه به جدول اضافه نمی شود.

```
--*****
go
set Xact_abort on;
go
begin transaction
insert into Tbl2 values (5)
insert into Tbl2 values (4)
insert into Tbl2 values (6)
commit transaction
go
select *
from Tbl2
```



نتیجه: هیچ کدام از رکورد ها به علت خطای ایجاد شده توسط رکورد شامل عدد ۴ به جدول اضافه نمی شوند.

Try...catch (۲۴-۲)

می توان ادعا کرد که برای هر بخش از کد این امکان متصور است که در زمان اجرا به مشکلی پیش بینی نشده برخورد کرده و خطایی را تولید کند. در بیشتر مواقع این موضوع که بتوان راهکاری برای به دام انداختن این خطا پیدا کرد، از اهمیت ویژه ای برخوردار است. همانطور که مشاهده شد، برای آزمایش هر خطا، از @@error استفاده کردیم که تعداد دستوراتمان را دو برابر می کند. واکنش بهتری نیز وجود دارد، به طوری که اگر یک مجموعه از عبارت ها، خطایی را در زمان اجرا تولید کرد، بتوان آنها را شناسایی و مهار نمود، اینجاست که ساختار TRY...CATCH مطرح می شود.

این ساختار شامل دو بخش می باشد بلاک TRY و بلاک CATCH. زمانی که یک خطا در یکی از عبارات T-SQL ای که داخل بلاک TRY قرار دارد تشخیص داده شود کنترل پاس داده می شود به بلاک CATCH جایی که خطا پردازش می شود. بعد از اینکه در بلاک CATCH خطا مدیریت شد کنترل منتقل می شود به اولین عبارت T-SQL که در ادامه ی عبارت END CATCH قرار دارد. در بلاک TRY عبارات T-SQL که بعد از عبارتی که موجب بروز خطا شد قرار دارند اجرا نخواهند شد. اگر هیچ خطایی داخل بلاک TRY وجود نداشته باشد کنترل پاس داده می شود به عبارتی که بلافاصله بعد از دستور END CATCH وجود دارد.

```
BEGIN TRY
    {T-SQL Statement}
END TRY
BEGIN CATCH
    {T-SQL Statement}
END CATCH
```



نکته: هر خطا دارای پارامتری به نام شدت خطا است، این ساختار خطاهایی با شدت ۱۰ و کمتر از آن را Handle نمی کند. به این معنا که اجرای دستورات داخل بلاک TRY ادامه پیدا میکنند و کنترل به بلاک CATCH منتقل نمی شود.



داخل محدوده ی بلاک CATCH توابع سیستمی زیر به جهت بدست آوردن اطلاعات پیرامون خطای رخ داده می توانند مورد استفاده قرار گیرند مثل شماره خطا، شدت خطا، پیام خطا و غیره.

```
ERROR_NUMBER(),
ERROR_SEVERITY(),
ERROR_STATE(),
ERROR_PROCEDURE(),
ERROR_LINE(),
ERROR_MESSAGE()
```



اولین تابع مشابه @@error شماره خطا را برمیگرداند.

چهارمین تابع از بالا، نام SP یا Trigger ای را که خطا در آن اتفاق افتاده است را برمیگرداند.

نکته : توابع فوق خارج از محدوده ی بلاک CATCH مقدار NULL را بر میگردانند.



متأسفانه تمام خطاها توسط این ساختار شناسایی نمی شوند. مثل خطای ارجاع معلق (Object Name Resolution). بطور نمونه می‌خواهیم داده های جدولی که وجود خارجی ندارد (قبلا ایجاد نشده است) را بازیابی کنیم.

```
begin try
    select *
    from NonExistentTable
end try
begin catch
    print 'error'
end catch
```



در اینجا کنترل به دست بلاک CATCH نمی افتد و پیغامی که مدنظر ما بود نمایش داده نمی شود.

مثال

```
BEGIN TRANSACTION;

BEGIN TRY
    insert into students ( id, FirstName, LastName, code )
    values ( 2, 2, 2, 2 )
COMMIT TRANSACTION;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS ErrorNumber,
        ERROR_SEVERITY() AS ErrorSeverity,
        ERROR_STATE() AS ErrorState,
        ERROR_PROCEDURE() AS ErrorProcedure,
        ERROR_LINE() AS ErrorLine,
        ERROR_MESSAGE() AS ErrorMessage;

    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
END CATCH;
GO
```



توضیح اجمالی اینکه در این مثال عبارت موجود در بلاک Try قیدی را نقض کرده در نتیجه کنترل به بلاک Catch منتقل می شود و در آنجا اطلاعاتی راجب خطای رخ داده صادر می شود و در صورتی که تراکنش فعالی وجود داشته باشد تراکنش Rollback می شود و کنترل اجرا برنامه به دستور بعد از END CATCH منتقل می شود و سپس برای اینکه تراکنشی که قبلا Rollback شده Commit نشود بررسی می کنیم که آیا تراکنش فعالی داریم یا خیر در صورتی که تراکنش فعالی داشتیم به این معناست که عبارات بدون خطا اجرا شدن پس تراکنش را Commit می کنیم.

نکته: بین Begin Tran و Begin try نباید دستور دیگری قرار بگیرد. همچنین اصولاً Begin tran داخل Begin try نوشته می شود.



مثال

BEGIN TRY

BEGIN TRANSACTION

```
insert into students ( id, FirstName, LastName, code )  
values ( 2, 2, 2, 2 )
```

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

```
SELECT ERROR_NUMBER () AS ErrorNumber
```

```
ROLLBACK TRANSACTION;
```

END CATCH

برای کشف خطا می توان از روش زیر نیز استفاده کرد.

بعضی از خطا ها همانطور که گفته شد به کمک دستورات بالا به دام نمی افتند. یکسری روال های تجربی برای کشف قسمتی از sp که خطا می دهد وجود دارد.

۱- ابتدا باید متوجه شویم sp در صورتی که پارامتر ورودی دارد با چه پارامتر هایی صدا زده شده ، برای این کار می توانیم از SQL Server Profiler استفاده کنیم.

در قسمت Tools\SQL Server Profiler را باید اجرا کنید و فیلتر هایی که لازم دارید اعمال می کنید و به کمک آن پارامترهای ورودی تنظیم شده برای sp مورد نظر را بدست می آورید.

۲- سپس اگر پارامتری در دستور If قرار دارد و باعث شده باشد که دستورات در بخش های مختلفی تقسیم شده باشند، با توجه به مقدار آن بررسی می کنیم که کدام دسته از دستورات اجرا خواهند شد.

```
if @status = 0
```

دستورات

```
else if status = 1
```

دستورات

```
else if status = 2
```

دستورات

```
else
```

دستورات



۳- بعد که یکسری دستورات پشت سر هم داریم می توانیم با دستور Print متوجه شویم کدام دستور خطا دارد.

```
if @status = 0
    دستورات
```

```
else if status = 1
    دستورات
```

```
else if status = 2
    1دستور
    2دستور
    print 2
    3دستور
    4دستور
    print 4
    5دستور
    print 5
else
    دستورات
```



فرض کنید 4 print اجرا می شود (در قسمت messages عدد ۴ را نمایش می دهد) و بلافاصله بعد از آن پیغام خطا نمایش داده می شود، در نتیجه متوجه می شویم که دستور ۵ خطا دارد.

۴- حال فرض کنید دستور ۵ یک select با کلی شرط های مختلف و sub query در قسمت های مختلف باشد. ابتدا باید متن خطا را متوجه شوید مثلاً وقتی more than one value باشد یعنی یک sub query که باید یک مقدار برگرداند دارد بیشتر از یک مقدار برگرداند. برای پیدا کردن اینکه کدام sub query باعث این خطا شده است می توانید بعضی از آنها را comment کنید و به جای آنها ۰ یا " برگردانید، در اینصورت یا خطا نمایش داده نمی شود که متوجه می شوید یکی از آنها خطا دارد و اصلاحش می کنید، و یا اینکه باز خطا دیده می شود در نتیجه متوجه می شوید sub query هایی که Comment کرده اید درست اند و آنهایی که comment نکردید مشکل دارند پس آنها را اصلاح می کنید.

تمرین



47- پروسیجر هایی که در تمرینات قسمت SP بود را با try...catch کامل کنید. (برای سوال ۴۲ انجام شود) (سوال * دار)

Index (۲۵)

Index این امکان را میدهد تا پایگاه داده راحت و سریعتر به داده ها دسترسی پیدا کند و نیازی به خواندن تمام اطلاعات برای پیدا کردن داده ای خاص نمی باشد. کاربر index را نمی بیند و فقط سرعت عملکرد را متوجه می شود.

نکته: دو نوع index وجود دارد Non-Clustered و Clustered .



وقتی یک داده دارای Clustered Index روی ستون فرضی A است، به این معنیست که داده ها روی دیسک بر اساس مقدار ستون A مرتب شده اند. برای مثال فرض کنید که یک کتابخانه داریم که تمام کتاب های آن بر اساس شماره نشر مرتب شده است. در صورتی که بخواهیم یک کتاب را جست و جو کنیم نیازی به پیمایش کل کتاب ها نیست و میتوانیم با استفاده از جست و جوی دودویی^۱ شماره نشر به سادگی به کتاب مورد نظر برسیم.

اما مشکل اینجاست که روی یک جدول، نمیتوان بیش از یک Clustered Index تعریف کرد، چرا که داده ها به صورت فیزیکی تنها میتوانند بر اساس یک ستون مرتب شوند. اما در صورتی که بخواهیم با استفاده از ستون دیگری نیز به جست و جو بپردازیم باید چه کار کنیم. مثلاً در مثال کتابخانه بخواهیم از روی نام کتاب به کتاب مورد نظر برسیم.

در این موارد میتوان از Non-Clustered Index استفاده نمود. به عبارت دیگر میتوان در جای دیگر، در یک لیست جداگانه لیست نام کتاب ها را به صورت مرتب نوشت و در کنار نام کتاب ها شماره نشر را یادداشت کرد. حال در صورتی که نام کتاب را داشته باشیم میتوانیم با استفاده از جست و جوی سریع شماره نشر را پیدا کرد و از روی شماره نشر به کتاب مورد نظر رسید. (بدیهی است که سرعت واکاوی اطلاعات از روی Clustered Index بالاتر است). البته میتوانستیم در لیست اضافه به جای شماره نشر، دقیقاً آدرس فیزیکی کتاب (قفسه دوم، ردیف سوم، کتاب هشتم) را یادداشت میکردیم، ولی توجه شود که در آن صورت با اضافه شدن یک کتاب ممکن بود مجبور به بروزرسانی کل آدرس های فیزیکی شویم.

در کل میتوان این نتیجه را گرفت، که تنها میتوان یک Clustered Index در یک جدول داشت اما میتوان بیشمار Non-Clustered Index در یک جدول تعریف کرد. مهمترین نکته انتخاب مناسب ستون ها برای ایجاد یک ایندکس جدید می باشد. ستون هایی که جست و جو روی آنها انجام میشود میتواند گزینه مناسبی برای تعریف Index باشد. اما توجه داشته باشید که به روز نگه داشتن این Index ها برای SQL هزینه بر است. و در صورتی که Index ها هوشمندانه تعریف نشود، تاثیر معکوسی داشته باشد یعنی باعث کاهش سرعت query ها شود.

از جمله کاندیدها برای تعریف ایندکس در یک جدول میتوان به موارد زیر اشاره کرد:

- ایندکس روی کلید خارجی،
- ایندکس روی ستون هایی که در Group By استفاده می شوند،
- ایندکس روی ستون هایی که در Oder By استفاده می شوند،
- ایندکس روی ستون هایی که در Where استفاده می شوند،
- ایندکس روی ستون هایی که در Union استفاده می شوند،
-

به صورت پیش فرض SQL با تعریف یک ستون به عنوان Primary key، یک Clustered Index برای آن کلید تعریف میکند، و باقی index هایی که ما تعریف میکنیم به صورت Non-Clustered تعریف میشود. اما این که کدام Index به عنوان Clustered تعریف شود توسط کاربر قابل تغییر است.

نکته: با استفاده از OR امکان استفاده از Index از بین می رود.



^۱ البته در واقع SQL Server از جست و جوی دودویی برای پیمایش روی ایندکسها استفاده نمیکند. ساختاری موثر تر به نام B-tree وجود دارد که از لحاظ کارایی میتواند بسیار موثرتر باشد. SQL از ساختار داده B-tree و B⁺-tree استفاده می کند.

نکته: برای مشاهده ی ایندکس های یک جدول می توان از Sp_help نیز استفاده کرد.



sp_help 'object name'



object name باید نام یک object در دیتابیس باشد مثلاً یک جدول یا پروسیجر و یا....

دستور فوق اطلاعات مربوط به object را بر می گرداند مثلاً برای یک جدول فیلد های آن ویا ایندکس های آن و بقیه اطلاعات مربوط به آن در دیتابیس را نمایش می دهد و یا مثلاً برای یک پروسیجر اطلاعات پارامتر های ورودی آن را نمایش می دهد.

با استفاده از این دستور دیگر لازم نیست در object explorer به دنبال object مورد نظر بگردید تا اطلاعات مربوط به آن را بدست آورید.

نکته: برای دیدن تعریف یک پروسیجر، تابع، trigger، computed column، check constraint، View نیز می توان از دستور sp_helptext استفاده کرد.



sp_helptext 'object name'



ابزارهایی وجود دارند که به ما در انتخاب ایندکس مناسب کمک می کنند: Database Engine Tuning Advisor و Actual Execution Plan

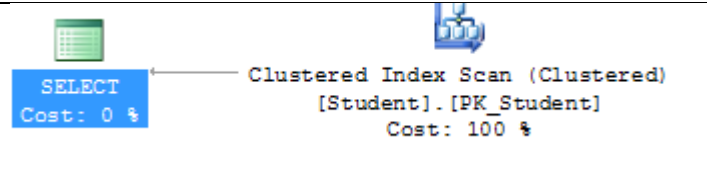
Database Engine Tuning Advisor (۲۴-۱)

این ابزار در قسمت Database Engine Tuning Advisor\ tools قرار دارد و با دریافت یک file یا table یا plan و cache و انجام تنظیمات مورد نظر یکسری ایندکس پیشنهاد می دهد که باید بررسی کنید و بعد در صورت مناسب بودن آن ایجادش کنید.

Actual Execution Plan (۲۴-۲)

این ابزار پرکاربردتر می باشد و با راست کلیک بر روی query می توانید آنرا انتخاب کنید، بعد از اجرای query در همان قسمتی که نتیجه و پیغام ها نمایش داده می شود یک tab جدید با عنوان execution plan نمایش داده می شود که هر دستور را بصورت جدا بررسی کرده و هزینه قسمت های مختلف آن را نمایش می دهد. شاید آنچه ما در Execution Plan مشاهده میکنیم با روندی که از اجرای دستور انتظار داریم کمی متفاوت باشد. این به دلیل تغییراتی است که بهینه کننده ی دستورات یا همان Optimizer روی پرس و جوها اعمال میکند تا سرعت اجرای دستورات را بیشتر نماید.

مثال : دانشجویانی با شماره دانشجویی ۸۸۰۰۱ را انتخاب کنید و Execution Plan این Query را مشاهده نمایید.

<pre>Select * from Student Where student.StudentCode = '88001'</pre>	
--	--

در این مثال یک عمل انجام شده است، و آن هم Clustered Index Scan بوده است. این عمل به معنای آن است که کل جدول Student اسکن (Scan) یا همان پیمایش شده است تا دانشجوی مربوطه انتخاب شود. ۱۰۰٪ هزینه‌ی انجام عملیات نیز مربوط به همین یک عمل است. حال میخواهیم که یک Index روی StudentCode یا همان شماره دانشجویی تعریف کنیم.

طریقه ساخت ایندکس به دو صورت است؛ با استفاده از کد:

```
create index studentIndex on dbo.students
```

(code)



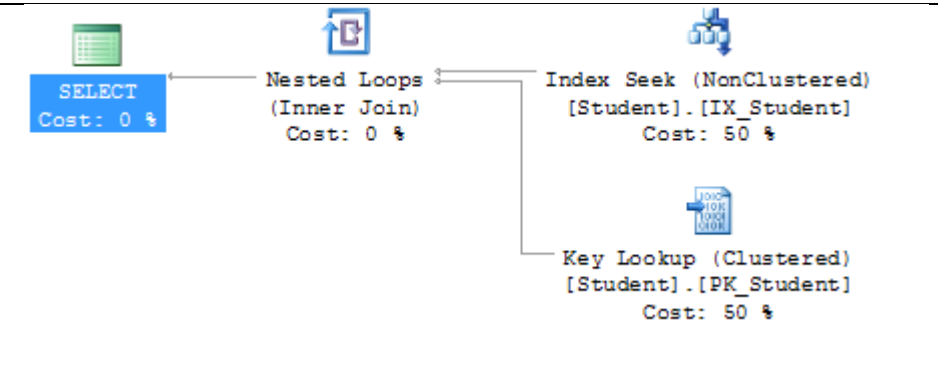
نکته: می توان چند ستون را با هم به عنوان یک ایندکس انتخاب کرد.



یا می توانید با راست کلیک روی indexes زیر هر جدول بصورت گرافیکی ایندکس را ایجاد کنید.

سوال: یک Non-Clustered Index روی ستون شماره دانشجویی از جدول دانشجو تعریف کنید.

حال دوباره میخواهیم دانشجویی با شماره دانشجویی ۸۸۰۰۱ را انتخاب و Execution Plan این Query را مشاهده نماییم.

<pre>Select * from Student Where student.StudentCode = '88001'</pre>	
--	--

مشاهده میکنید که این بار روش اجرای دستور متفاوت شده است. در ابتدا عمل Index Seek روی همان Non-Clustered Index تعریف شده انجام شده است و با استفاده از آن کلید اصلی دانشجوی مورد نظر پیدا شده است. پس از آن از روی Clustere Index یک عمل KeyLookup انجام شده است و دانشجویی با کلید اصلی مورد نظر پیدا شده است. در واقع Execution Plan روش اجرای هر دستور را به ما نشان میدهد و علاوه بر پیشمار اطلاعات مفیدی که ارائه میدهد، از روی هزینه هایی که اعلام می کند می توان فهمید کدام ایندکس هزینه ی زیادی داشته و در واقع برعکس دارد عمل می کند. می توان در صورت لزوم آن را حذف کرد. همچنین فلشهای

روی تصویر، جابه‌جایی داده روی عملگرها را مشخص میکند (هرچه یک فلش بزرگتر باشد نشان دهنده‌ی تعداد سطرهای داده‌ی بیشتر منتقل شده به عملگر بعدی است).

نکاتی درباره Execution Plan	
✓	با نگه داشتن Mouse روی هر عملگر یا فلش میتوان اطلاعات دقیق‌تری از هر عملگر به دست آورد.
✓	با RightClick روی هر آیتم گرافیکی و انتخاب Properties میتوان تمام اطلاعات هر عملگر را مشاهده کرد.
✓	خواندن کامل یک Execution Plan یک تخصص است و نیاز به تمرین و مطالعه فراوان دارد.
✓	یک Execution Plan میتواند به صورتهای متفاوتی دیده شود از جمله: <ul style="list-style-type: none"> ○ گرافیکی ○ متنی ○ XML
✓	میتوان روی یک Execution Plan، راست کلیک کرد و آن را به صورت یک فایل XML ذخیره کرد. این فایل قابلیت ارسال به دیگر کاربران جهت آنالیز را دارد. با باز کردن این فایل میتوان در هر زمانی این Execution Plan را دوباره مشاهده کرد.

دستور حذف ایندکس:

DROP INDEX [studentIndex] ON [dbo].[Students]



نکته: می توان از where در تعریف index استفاده کرد.(Filtered index)



create index test

on sct (student_id, course_id, teacher_id)

where score is not null



Back up-Restore (۲۶)

برای گرفتن نسخه پشتیبان (Back up) از روی یک دیتابیس می توان یکی از راه های زیر را با توجه به نیازها انتخاب کرد.

- راست کلیک روی دیتابیس و انتخاب tasks\back up
- استفاده از منوی management\maintenance plan

در صورتی که بخواهیم در بازه زمانی های مشخص شده از روی دیتابیس نسخه پشتیبان تهیه کنیم بهتر است از روش دوم استفاده کنیم. روی maintenance plan راست کلیک کرده و گزینه ی maintenance plan wizard را انتخاب می کنیم بعد از نام گذاری در قسمت schedule دکمه ی change را می زنیم و تنظیمات مورد نظر را اعمال می کنیم. بعد از آن نوع back up گیری را مشخص می کنیم.

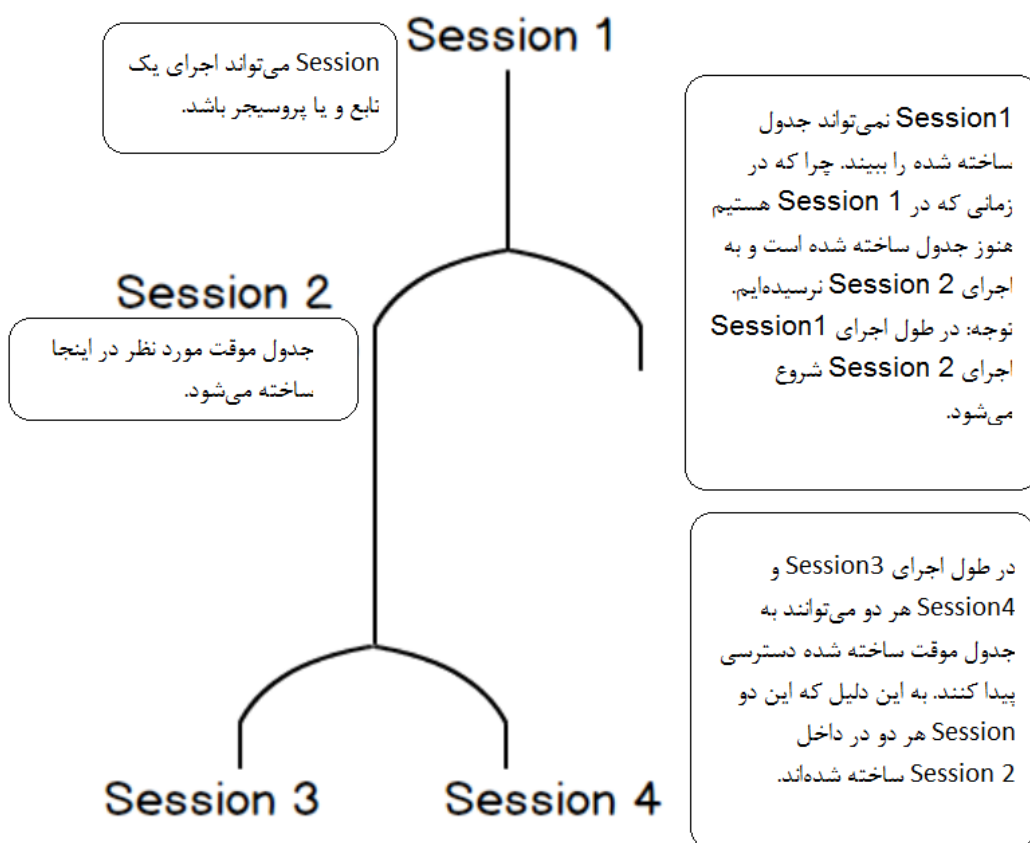
برای بازخوانی نسخه پشتیبان باید آن را Restore کنیم، برای این کار روی databases راست کلیک کرده و گزینه ی Restore files and filegroups را انتخاب می کنیم. سپس اگر دیتابیس وجود داشته باشد آن را انتخاب کرده و گزینه نام دیتابیس جدید را وارد می کنیم. و بعد از آن فایل نسخه پشتیبان را انتخاب می کنیم.

موفق باشید

ضمیمه A: جداول موقت یا Temporary Tables

در SQL Server، جداول موقت در زمان اجرا ایجاد می شوند و شما می توانید تمام عملیات را که می توانید در یک جدول معمول انجام دهید انجام دهید. این جداول در داخل پایگاه داده Tempdb ایجاد شده است. بر اساس دامنه و رفتار جدول های موقت از دو نوع به عنوان زیر ارائه شده است:

- جداول موقت محلی و یا Local Temp Table
- Local temp table فقط برای سازنده آن table در دسترس است (single user)
- فقط در Session ای که ساخته شده و تمام Sub Session های آن دیده خواهد شد.
- به صورت خودکار وقتی Session ای که در آن ساخته شده را ببندیم، از بین خواهد رفت.
- اسم Local temporary table با علامت «#» نشان داده می شود.
- در لیست Table ها نشان داده نمی شود.
- یکی از استفاده های مهم این جداول پیاده سازی حلقه هاست. استفاده دیگر برای نگه داری نتایج موقت، و جلوگیری از محاسبه دوباره آنهاست.
- یک تمرین برنامه نویسی خوب آن است در کد، زمانی که کارمان با یک جدول موقت تمام شد، آن را با دستور Drop حذف نماییم.



جداول موقت سراسری Global Temp Table

مهمترین تفاوت آنها با جداول موقت محلی آن است که زمانی که این جداول ساخته می شوند، در تمام Session های دیگر، اعم از آنکه زیر مجموعه Session سازنده باشد و یا نباشد، جدول موقت را خواهد دید و میتواند در آن به Insert، Update، و یا Delete بپردازد.

عمر این جداول نیز با پایان یافتن Session سازنده پایان می یابد.

متغیر از نوع جدول و یا Table Variable

Table Variable ها به شکل متغیر تعریف می شوند، اما متغیری از جنس جدول.

Table Variable ها صرفاً به ازای Session ایجاد کننده در دسترس هستند. (دقت شود که بر خلاف جداول موقت، حتی در Sub Session ها نیز قابل دسترس نیستند، اما میتوان آنها را به عنوان آرگمان به توابع دیگر پاس کرد)

طول عمر Table Variable ها از نقطه تعریف در Session تا جایی که Session پایان می یابد است.

اسکرپت ایجاد Table Variable

```
DECLARE @Table TABLE(  
    name varchar(30) NOT NULL,  
    location varchar(30) NOT NULL  
);
```

```
INSERT @Table  
SELECT name, location  
FROM Exec SPROC @param , @param
```

