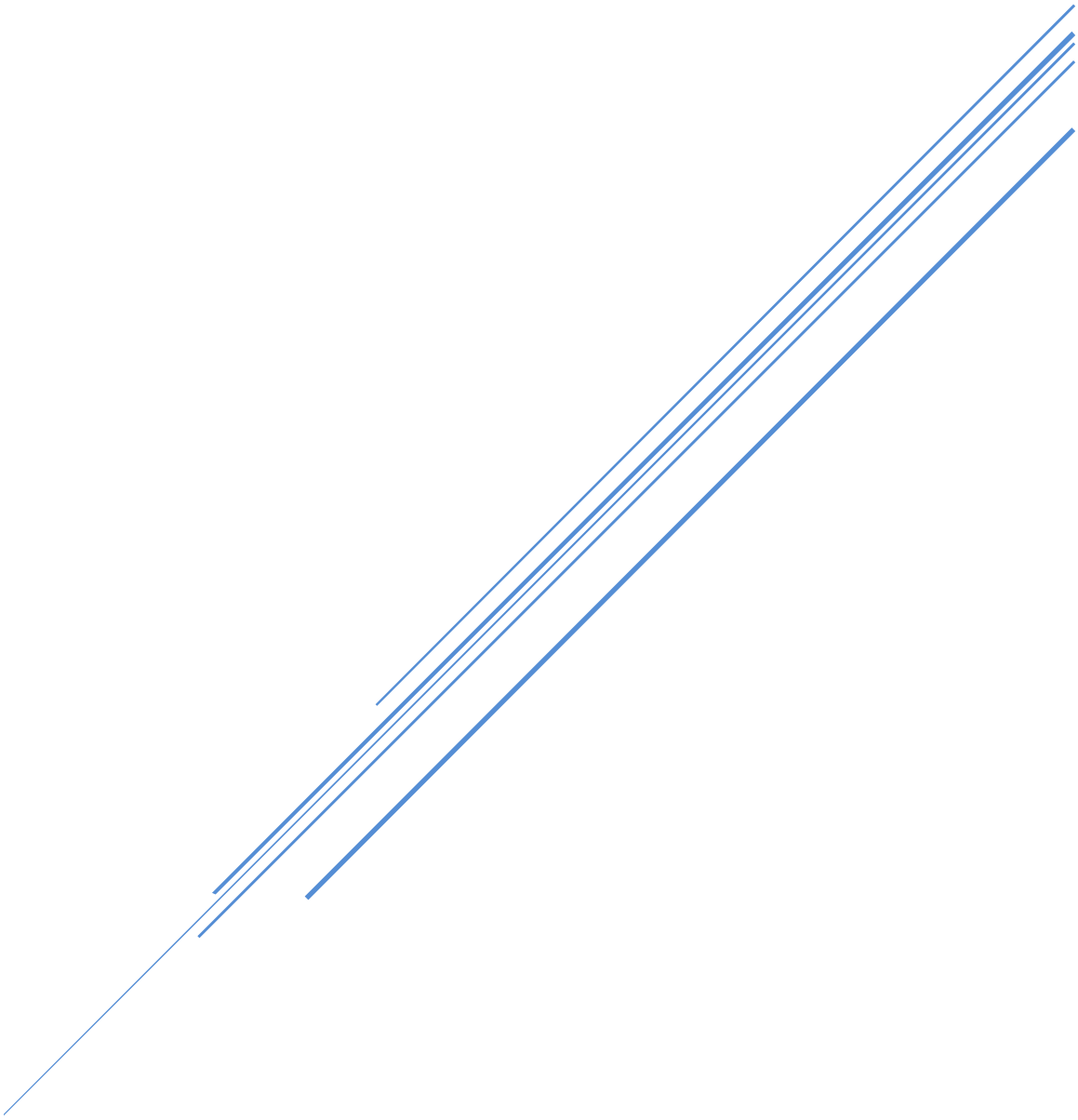
آموزش کاربردی

MICROSOFT SQL SERVER 2008

محمد امین ابوالحسن نژاد



فصل اول : مقدمه ای بر سیستم های پایگاه داده رابطه ای

- **مروری بر سیستم های پایگاه داده :**

یک سیستم پایگاه داده ، مجموعه ای از اجزای نرم افزاری و پایگاه داده های مختلف است که از بخش های زیر تشکیل شده است :

- **برنامه های کاربردی :** نرم افزارهای خاصی هستند که توسط کاربران یا شرکت های نرم افزاری طراحی می شوند.
- **اجزای سرویس گیرنده:** نرم افزارهای خاصی هستند که توسط شرکت های متخصص پایگاه داده طراحی و تولید می شوند و کاربران توسط اجزای سرویس گیرنده به داده های ذخیره شده دستیابی پیدا می کنند.
- **سرور :** برای مدیریت داده های ذخیره شده روی پایگاه داده کاربرد دارد . (هر سرویس گیرنده از طریق ارسال یک پرس و جو ، با سرور ارتباط برقرار می کند، پرس و جو توسط سرور بررسی و سپس نتیجه به سرویس گیرنده ارسال می شود).
- **پایگاه داده :** این بخش از پایگاه داده از دو دید بررسی می شود : ۱- دید کاربران ۲- دید سیستم پایگاه داده.

از دید کاربران ، پایگاه داده مجموعه ای داده های منطقی مرتبط با هم است . و از دید سیستم پایگاه داده ، پایگاه داده دنباله ای بایت هاست که روی دیسک ذخیره شده است .

- **هر سیستم پایگاه داده باید دارای یک سری ویژگی هایی باشد :**
  - **واسط کاربری** که شامل منوها ، فرم ها و ... می باشد.
  - **استقلال فیزیکی داده ها** بدین معناست که برنامه های کاربردی به ساختار فیزیکی داده های ذخیره شده در پایگاه داده وابسته نیستند .
  - **استقلال منطقی داده ها** بدین معناست که با هر تغییری در ساختار فایل همه برنامه های کاربردی مورد نظر تغییر پیدا می کنند.
  - **بهینه سازی پرس و جو** بدین معناست که هر پایگاه داده باید دارای یک بهینه ساز (Optimizer) باشد تا از بین پرس و جو های ممکن بهترین را انتخاب نماید .
  - **جامعیت داده ها** یعنی این که داده های ناسازگار را تشخیص و از ذخیره آن ها جلوگیری کند .مثلا ۳۰ مارس و ساعت ۸.۵۰ دقیقه رابطه ای با هم ندارند پس باید از ذخیره آن جلوگیری کند.

- **کنترل هم روندی** یعنی این که اگر برنامه های کاربردی مختلف سعی داشتند داده های یکسانی را به روز کنند ، تغییرات به صورت کنترل شده باشد . برای مثال دو نفر داریم که صاحب یک حساب بانکی مشترک هستند . موجودی حساب ۲۰۰۰۰ تومان است . حال اگر این دو هم زمان به بانک مراجعه کنند و هرکدام ۱۰۰۰۰ برداشت نمایند باید موجودی پس از دریافت وجه صفر باشد .
- **پشتیبان گیری و ترمیم**
- **امنیت**

مدل داده ای : مجموعه ای از مفاهیم ، روابط بین آن ها و محدودیت هایی است برای نمایش داده های یک مسئله واقعی.

مثال : شرکتی را در نظر بگیرید که دارای یک سری کارمند ، اداره می باشد و هر کارمند متعلق به یک اداره است و هر کارمند می تواند در چند پروژه شرکت داشته باشد :

department(dept_no,dept_name,dept_location)

هر اداره دارای یک شماره شناسایی ، یک نام و یک محل است . (معمولا ما در جدول های دیگر با dept_no کار می کنیم .)

employee(emp_no,emp_fname,emp_lname,dept_no)

هر کارمند هم دارای شماره شناسایی ، نام و نام خانوادگی است و در آخر متعلق به کدام اداره است .

project(prj_no,prj_name,budget)

هر پروژه در شرکت با یک نام ، شماره شناسایی و بودجه معرفی می شود.

work_on(emp_no,prj_no,job,date)

و در آخر کارمندی با شماره شناسایی (مثلا ۱۲۳۴۵۶) ، در یک پروژه شرکت دارد و کار و تاریخ آن هم مشخص است .

جدول ۱ – department

dept_no	dept_name	dept_location
100	اداره تحقیقات	مشهد
101	اداره حسابداری	تهران
102	اداره فروش	بیرجند

شماره شناسایی باید منحصر به فرد باشد یعنی نباید دو اداره دارای دو شماره شناسایی یکسان باشند.

جدول ۲ – employee

emp_no	emp_fname	emp_lname	dept_no
9201	محمد	محمدی	100
9202	حسن	حسنی	100
9203	علی	علوی	101
9204	محسن	محسنی	102

شماره شناسایی هر کارمند باید منحصر به فرد باشد (مثل کارت ملی هر فرد). و هم چنین باید مشخص کرد که هر کارمند در چه اداره ای مشغول به کار است .

جدول ۳– Project

prj_no	prj_name	budget
1	پروژه ساخت گوشی های همراه	1000,000,000
2	پروژه ساختمان مرکزی	150,000,000

جدول ۴– Work_on

emp_no	prj_no	job	date
9201	2	آنالیزور	1392/02/01
9201	1	آنالیزور	1392/04/08
9203	1	دفتر دار	1392/12/12

برای مثال از جدول Work_on می توانیم متوجه شویم که آقای محمد محمدی که کارمند اداره تحقیقات است در پروژه های ساخت گوشی های همراه و ساختمان مرکزی به عنوان آنالیزور مشغول به کار بوده است .

- **SQL** :

زبان پایگاه داده رابطه ای SQL Server ؛ Transact_sql است . اس کیو ال یک زبان مجموعه گراست یعنی می تواند سطر های زیادی از یک یا چند جدول را توسط یک دستور به دست آورد. هم چنین یک زبان غیر رویه ای هم می باشد.

SQL دو زبان دارد : ۱- DML (Data Manipulation Language) : زبان دستکاری داده ها که برای دستکاری داده ها استفاده می شود و شامل دستورات : Select ، Insert ، Delete و Update می باشد.

۲- DDL(Data Definition Language) : زبان تعریف داده ها که برای کار با شمای جداول پایگاه داده است که شامل دستورات Create، Alter و Drop است .

- **آشنایی با برخی مفاهیم :**
  - **نرمال سازی :** پروسه یافتن وابستگی بین ستون های یک جدول . در صورت پیدا شدن ، جدول به دو جدول تقسیم می شود و این پروسه ادامه دارد تا زمانی که وابستگی بین ستون های یک جدول نباشد.
  - **وابستگی تابعی :** به ازای هر مقدار از یک ستون ، فقط یک مقدار منحصر به فرد در ستون دیگر وجود داشته باشد . برای مثال به ازای شماره شناسایی ۹۲۰۱ ، فقط یک نفر با یک نام خانوادگی وجود دارد.
  - **وابستگی چند مقداری :** به ازای یک مقدار از یک ستون ، مجموعه ای مقادیر داشته باشیم برای مثال یک نویسنده می تواند چندین کتاب داشته باشد.
  - **خصوصیت مرکب :** یعنی حاوی چند مقدار اتمیک (چند مقداری) است . مثل آدرس که شامل شهر ، خیابان ، کوچه و ... است .
  - **خصوصیت مشتق :** یعنی مقدارش با محاسبه مقادیر ستون های دیگر به دست می آید . مثل average معدل دانشجویان یک کلاس.

- **فرم نرمال سطح یک (1NF)** : در هر ستون جدول فقط یک مقدار می تواند ذخیره باشد .

emp_no	prj_no	job	Date
9201	1,2	آنالیزور	...

این جدول در سطح نرمال یک قرار ندارد چون ، در ستون Prj_no دو مقدار ذخیره شده است :

emp_no	prj_no	job	Date
9201	1	آنالیزور	...
9201	2	آنالیزور	...

- **فرم نرمال سطح دو (2NF)** : جدولی 2NF است که اولا 1NF باشد و ثانیا هیچ ستون غیر کلیدی به کلید اصلی جدول وابسته نباشد .

emp_no	prj_no	job	date	dept_no
9201	1	آنالیزوز	...	100
9201	2	آنالیزوز	...	100

جدول بالا در سطح 2NF نیست چون کلید های اصلی این جدول emp_no و prj_no هستند و dept_no به emp_no وابستگی دارد و باید حذف شود.

نکته : هر جدولی که کلید اصلی آن یک ستون باشد ، همیشه در سطح 2NF است .

- **فرم نرمال سطح سه (3NF)** : جدولی 3NF است که اولا 2NF باشد و ثانیا هیچ وابستگی تابعی بین ستون های غیر کلیدی وجود نداشته باشد .

emp_no	emp_fname	emp_lname	dept_no	dept_name
9201	محمد	محمدی	100	اداره تحقیقات
...	...	...	...	...

جدول بالا 3NF نیست چون وابستگی تابعی بین dept_no و dept_name وجود دارد .

فصل دوم – اجزای SQL

زبان Database Engine دارای یک سری ویژگی های اساسی مانند زبان های برنامه نویسی دیگر است : مقادیر حرفی ، جداکننده ها ، توضیحات ، شناسه ها و کلمات کلیدی .

- **مقادیر حرفی :**

هر مقدار حرفی می تواند به صورت الفبایی ، مبنای شانزده یا عدد باشد. هر رشته داخل یک جفت تک کوتیشن (' ') یا یک جفت دابل کوتیشن (" ") قرار می گیرد. برای نمایش یک تک کوتیشن به عنوان رشته کافی است از دو تک کوتیشن استفاده شود .

- **جداکننده ها :**

یکی از استفاده های دیگر دابل کوتیشن (" ") در جداکردن شناسه هاست.

- **توضیحات :**

دو روش برای بیان توضیحات در SQL وجود دارد . زمانی که بخواهید یک خط توضیح را بیان کنید از (--) استفاده می شود و هنگامی که بخواهید چندین سطر توضیح بیان کنید از (/* */) استفاده می شود.

مثال : بیان توضیح تک خطی و چند سطری :

```
-- Microsoft SQL Server 2008 R2

/*  Microsoft SQL Server
2008
R2
*/
```

- **شناسه ها :**

برای مشخص کردن پایگاه داده ها ، جداول و اندیس ها و ... از شناسه ها استفاده می شود. شناسه ها رشته های کارکتری به طول حداکثر ۱۲۸ کارکتر هستند. مثلا کارکتر @ در ابتدای شناسه ، نشان دهنده متغیر است .

- **نوع داده ها**

T-SQl از نوع داده های مختلفی استفاده می کند :

- عددی
- کارکتری
- زمانی
- گوناگون

- **نوع داده عددی :**

نوع داده	شرح
int	برای ذخیره مقادیر صحیح در ۴ بایت
Smallint	برای ذخیره مقادیر صحیح در ۲ بایت (از 32768- تا 32768)
tinyint	برای ذخیره اعداد صحیح غیر منفی در ۲ بایت (بین ۵ تا ۲۵۵)
bigint	برای ذخیره اعداد صحیح در ۸ بایت
decimal(p,s)	برای ذخیره اعداد اعشاری که p تعداد کل ارقام به همراه ارقام سمت راست نقطه اعشار. (متناسب با مقدار p بین ۵ تا ۱۷ بایت ذخیره می شوند.)
Numeric	همانند decimal می باشد.
Real	برای ذخیره مقادیر اعشاری شناور
Float[p]	برای ذخیره اعداد اعشاری شناور با دقت p که اگر p>25 باشد ۸ بایت در نظر گرفته می شود و در غیر این صورت ۴ بایت .
Money	برای ذخیره مقادیر پولی ، ۸ بایتی بوده و با ۴ رقم اعشار گرد می شود.
SmallMoney	برای ذخیره مقادیر پولی اما در ۴ بایت.

- **نوع داده های کارکتری :**

نوع داده	شرح
char[(n)]	برای ذخیره یک رشته با طول ثابت که n تعداد کارکتر ها را نشان می دهد. (در صورتی که n را ۱۰ قرار داده و کمتر از ۱۰ کارکتر وارد کنید ، مقادیر باقی مانده را فضای خالی در نظر می گیرد.) n می تواند حداکثر ۸۰۰۰ باشد .
nchar[(n)]	برای ذخیره یک رشته با کارکترهای یونیکد به طول n که حداکثر n می تواند ۴۰۰۰ باشد.
varchar(n)	همانند char است ولی با این تفاوت که اگر n را ۱۰ در نظر بگیرید و ۵ کارکتر وارد کنید ، باقی را فضای خالی در نظر نمی گیرد).
nvarchar(n)	همانند varchar اما با این تفاوت می تواند مقادیر یونیکد را دریافت کند.

مثال : اگر دو فیلد با نام های a و b که نوع اول char و نوع دومی nvarchar باشد را تعریف کنیم و در هر دو فیلد عبارت (سلام) را وارد کنید، پس از بازخوانی در فیلد a به جای سلام ؟؟؟؟ را نمایش خواهد داد.

- **نوع داده های زمانی :**

نوع داده	شرح
date	برای ذخیره تاریخ در ۳ بایت
time	برای ذخیره زمان در ۳ تا ۵ بایت
DateTime	برای ذخیره زمان و تاریخ در ۴ بایت
SmallDateTime	برای ذخیره زمان و تاریخ در ۲ بایت
DateTime2	برای ذخیره زمان و تاریخ با دقت بالا در ۶ تا ۸ بایت و دقت زمان ۱۰۰ نانو ثانیه است.
DateTimeOffset	برای ذخیره زمان و تاریخ و هم چنین برای ذخیره منطقه زمانی در ۶ تا ۸ بایت

- **نوع داده های Text ، nText و Image**:

برای ذخیره متون قابل چاپ از text و ntext و برای ذخیره داده های ماژولی ، صوت / تصویر از Image استفاده می شود. نوع داده image به صورت ساختار b-tree ذخیره می شود.

- **نوع داده Uniqueidentifier**:

برای ذخیره یک عدد شناسایی منحصر به فرد ۱۶ بایتی به کار می رود.

- **نوع داده Sql_Variant**:

برای ذخیره داده هایی با نوع مختلف به کار می آید.

- **توابع در T-SQL**:
  - تجمعی
  - اسکالر
  - جدولی

- **توابع تجمعی :**

توابع تجمعی به یک گروه از مقادیر یک ستون اعمال می شود و همواره یک مقدار را بر می گرداند.

- توابع تجمعی متداول
- آماری
- کاربری
- تحلیلی

- **توابع تجمعی :**
  - AVG : میانگین ریاضی مقادیر یک ستون را محاسبه می کند.
  - MAX-MIN : مقدار حداقل و حداکثر مقادیر یک ستون را برمی گرداند.
  - SUM : مجموع مقادیر یک ستون را برمی گرداند.
  - COUNT : تعداد مقادیر غیر null یک ستون را برمی گرداند و خروجی آن از نوع int است.
  - BIGCOUNT : مانند COUNT است ولی مقدار خروجی از نوع bigint است.

- **توابع اسکالر :**

روی مقدار یا لیستی از مقادیر عمل کرده و یک مقدار منفرد برمی گرداند.

- عددی
- تاریخ
- رشته ای
- سیستمی
- متاداده

- **توابع عددی :**

تابع	شرح
ABS(n)	قدرمطلق یک عدد را برمی گرداند. `select ABS(-2)`
Acos(n)	آرک کسینوس یک عدد را برمی گرداند و خروجی از نوع float است. `select ACOS(1)`
Asin(n)	آرک سیسنوس یک عدد را برمی گرداند.
Atan(n)	آرک تانژانت یک عدد را برمی گرداند.
Atn2(n)	آرک کتانژانت را بر می گرداند.
Sin(n)	مقدار سینوس عدد را برمی گرداند
Cos(n)	مقدار کسینوس عدد را برمی گرداند
Tan(n)	مقدار تانژانت عدد را برمی گرداند

Cot(n)	مقدار کتانژانت عدد را برمی گرداند
Ceiling	بزرگترین عدد را برمی گرداند. select CEILING(4.1) → 5 select CEILING(-4.1) → -4
Degrees(n)	رادیان را به درجه تبدیل می کند. select DEGREES(PI()/2) → 90
Floor(n)	کف عدد ورودی را برمی گرداند. select Floor(PI()) → 3
Radians(n)	درجه را به رادیان تبدیل می کند. select RADIANS(180)→ 3
Sqrt(n)	جذر عدد ورودی را می گیرد. select SQRT(16)→ 16
Square(n)	عدد ورودی را به توان دو می رساند. select SQUARE(3) → 9
Log10(n)	لگاریتم عددی ورودی را با پایه ۱۰ برمی گرداند. select LOG10(100) → 2
Power(x,y)	x^y را محاسبه می کند. select POWER(10,2.5) → 316

- **توابع تاریخ**

تابع	شرح
GetDate()	تاریخ و زمان جاری سیستم را برمی گرداند. select GETDATE() → 2013-11-01 20:26:10.100
DateName(item,date)	select DATENAME(SECOND,getdate() → 10 select DATENAME(MINUTE,getdate()) → 26 select DATENAME(HOUR,getdate()) → 20 select DATENAME(WEEKDAY,getdate()) → Friday select DATENAME(MONTH,getdate()) →November select DATENAME(YEAR,getdate()) → 2013 select DATENAME(DAYOFYEAR,getdate()) → 305 select DATENAME(WEEK,getdate()) → 44
DateDiff(item,date1,date2)	اختلاف بین دو تاریخ را برمی گرداند. select DATEDIFF(SECOND,'2013-11-01 20:40:03.750','2013-11-01 20:41:20.653')→ 77 select DATEDIFF(MILLISECOND,'2013-11-01 20:40:03.750','2013-11-01 20:41:20.653')→76909 select DATEDIFF(MINUTE,'2013-11-01 20:40:03.750','2013-11-01 20:41:20.653') → 1

select DATEDIFF(HOUR,'2013-11-01 20:40:03.750','2013-11-01 20:41:20.653') → 0	
روز هفته از تاریخ جاری سیستم را برمی گرداند. select DATEPART(WEEKDAY,GETDATE())→ 6	DatePart(item,date)

مثال : چاپ سال شمسی از روی تاریخ میلادی :

```
declare @MiladiYear int;
declare @ShamsiYear int;

set @MiladiYear =  DATEPART(YEAR,getdate());
print ('Miladi : '+CAST(@MiladiYear as char))
set @ShamsiYear=@MiladiYear-621;
print ('Shamsi : '+CAST(@ShamsiYear as char))
```

```
Miladi : 2013
Shamsi : 1392
```

توضیح : ابتدا با استفاده از کلمه کلیدی declare ، متغیری را تعریف می کنیم. (declare @var type ؛ به جای var نام متغیر را وارد کرده و جای type ، نوع متغیر را وارد می کنیم .) برای مقداردهی متغیرها از کلمه کلیدی Set کمک می گیریم.

توضیحی راجع به CAST که برای تبدیل استفاده می شود. CAST(@var as type)

- **توابع رشته ای**

تابع	شرح
Ascii(character)	کد اسکی کارکتر ورودی را برمی گرداند. select ASCII('a') → 97
Char(n)	کد وارد شده را به کارکتر معادل تبدیل می کند. select CHAR(97)→ a
Charindex(c1,c2)	موقعیت c1 را در c2 جستجو کرده و موقعیت شروع آن را بر می گرداند. select CHARINDEX('amin','that boy is amin') → 13
Left(c1,i1)	به مقدار i1 از c1 شروع به نمایش می کند. select LEFT('that boy is amin',4) → that
Len(character)	تعداد کارکتر های وارد شده را برمی گرداند. اگر فضای خالی بعد از رشته باشد ، آن ها را درنظر نمی گیرد . اما قبل از رشته اگر فضای خالی ای وجود داشته باشد ، آنها را در نظر می گیرد. select LEN('amin')→ 4 select LEN('amin ')→ 4

select LEN(' amin') → 6 select LEN('amin ali') → 8	
کارکتر های بزرگ را به کارکترهای کوچک تبدیل می کند. select LOWER('Ali') → ali	Lower(character)
کارکتر های کوچک را به کارکترهای بزرگ تبدیل می کند. select Upper('Ali') → ALI	Upper
فضاهای خالی سمت چپ را حذف می کند. select LTRIM(' amin') → amin	Ltrim
فضاهای خالی سمت راست را حذف می کند.	Rtrim
برای برعکس کردن کارکتر مورد استفاده قرار می گیرد. select REVERSE('ab') → ba	Reverse

- **توابع سیستمی :**

تابع	شرح
Cast(a as type)	برای تبدیل عبارت a به نوع مورد نظر کاربرد دارد. declare @a int set @a=20 print('char a ='+cast(@a as char))
Current_user	نام کاربر جاری را برمی گرداند. select current_user → dbo
User	نام کاربر جاری را برمی گرداند.
User_ID	شناسه کاربر وارد شده را برمی گرداند. select USER_ID('dbo') → 1
User_Name	با وارد کردن شناسه کاربر ، نام کاربر را برمی گرداند. select USER_NAME(1)→ dbo
System_user	کاربر جاری سیستم را برمی گرداند. select SYSTEM_USER → AMIN\ASUS-K50IN
IsNumberic	بررسی می کند که آیا عدد ورودی تابع ، عددی معتبر است یا خیر.

- **توابع متاداده**

تابع	شرح
db_name	نام پایگاه داده را باتوجه به شناسه ورودی برمی گرداند. select DB_NAME(1) → master

db_id	شناسه پایگاه داده ورودی را برمی گرداند. `select DB_ID('master') → 1`

- **متغیر های سراسری**

متغیر	شرح
@@ language	زبان جاری به کار رفته در پایگاه داده را نشان می دهد. `select @@LANGUAGE 1 us_english`
@@langid	شناسه زبان جاری به کار رفته در پایگاه داده را نشان می دهد. `select @@LANGID`
@@Version	نسخه جاری نرم افزار سیستم پایگاه داده را برمی گرداند. `select @@VERSION → Microsoft SQL Server 2008 R2 (RTM) - 10.50.1600.1 (X64) Apr 2 2010 15:48:46 Copyright (c) Microsoft Corporation Enterprise Edition (64-bit) on Windows NT 6.1 <X64> (Build 7600: Service Pack 1)`
@@servername	اطلاعاتی درباره سرور پایگاه داده محلی برمی گرداند. `select @@SERVERNAME → AMIN`
@@ServiceName	نام سرویسی که از آن استفاده می کنیم را نشان می دهد. `select @@SERVICENAME → MSSQLSERVER`

تمرین

۱- فرق بین داده های int , smallint,tinyint چیست ؟

۲- تفاوت بین داده های Char و varchar چیست ؟

۳- چگونه می توان فرمت یک ستون را به صورت yyyy/mm/dd تنظیم کرد؟

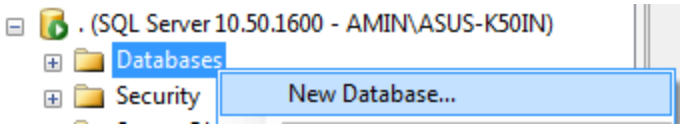
فصل سوم – زبان تعریف داده (DDL)

همانطور که قبلا بیان شد ، Sql دارای دو زبان می باشد : DML و DDL که DDL برای کار با شمای پایگاه داده هاست .

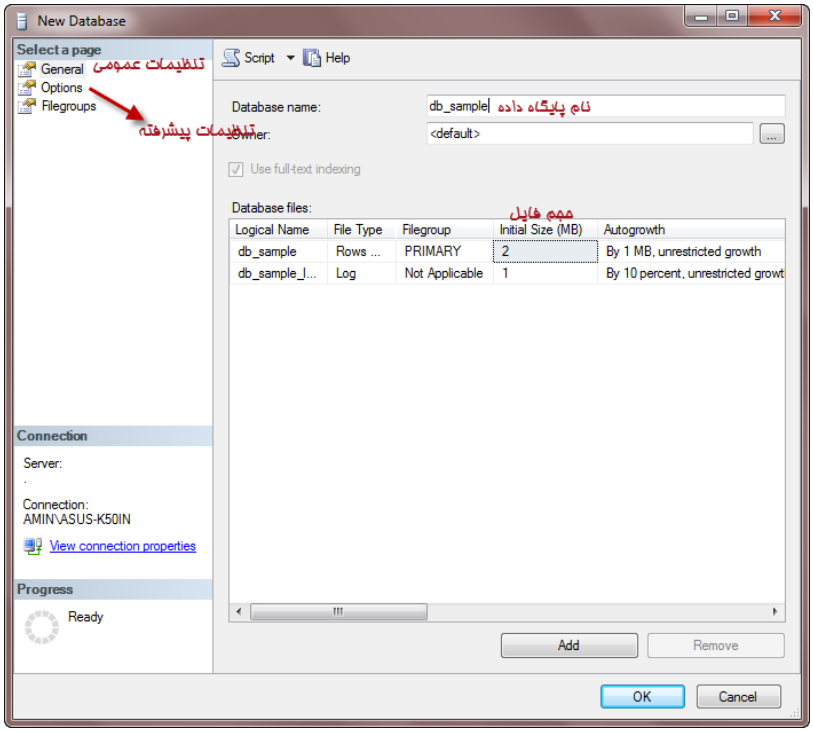
- **ایجاد اشیای پایگاه داده :**

اولین شی از پایگاه داده که قبل از بقیه اشیا باید ایجاد شود ، خود پایگاه داده است . پایگاه داده های کاربری و سیستمی توسط Database Engine مدیریت می شوند. یک کاربر مجاز می تواند یک پایگاه داده ایجاد کند ، درحالی که پایگاه داده های سیستمی در حین نصب ایجاد می شوند و عبارتند از : master ، model ، tempdb ، msdb و resource .

برای ایجاد پایگاه داده دو روش وجود دارد : روش اول از طریق رابط کاربری و راه دیگر توسط T-sql .



پس از کلیک راست بر روی database و انتخاب گزینه New Database... صفحه ای به صورت زیر باز می شود:



ایجاد پایگاه داده با استفاده از دستورات T-sql :

- ایجاد یک پایگاه داده با نام db_sample :

```
use master
go
create database db_Sample   // حداکثر نام می تواند ۱۲۸ کارکتری باشد.
go
```

حداکثر ۳۲۷۶۷ پایگاه داده ، توسط سیستم مدیریت می شود.

- حذف پایگاه داده db_sample

```
use master
go
drop database db_Sample
go
```

- ایجاد پایگاه داده با نام db_sample با تنظیمات پیشرفته تر :

```
use master
go
create database db_Sample
on(Name='db_Sample_data' , filename='d:\sql\db_Sample_data.mdf',
size=5mb,maxsize=150mb,FileGrowth=15%)
log on(Name='db_Sample_Code',filename='d:\sql\db_Sample_code.ldf')
go
```

فایل db_sample_data ، به عنوان فایل primary در نظر گرفته شده است . حال اگر چند فایل داشته باشیم، معمولا فایل اول به عنوان فایل اصلی در نظر گرفته می شود. [اگر بخواهیم ، فایلی را به عنوان فایل اصلی در نظر بگیریم ، کلمه کلیدی primary را درنظر می گیریم.]

در قسمت name ، باید نام فایل mdf ، در بخش filename باید مسیر فایل mdf ، در بخش size اندازه فایل ، در بخش maxsize اندازه نهایی فایل و در بخش filegrowth میزان رشد فایل را می توان مشخص نمود که در filegrwoth می توان به دو صورت اندازه افزایشی فایل را مشخص کرد یا به صورت عددی (KB,MG,GB,TB) و یا به صورت درصدی .

در قسمت maxsize ، اگر مقدار آن را مشخص نکنید یا Unlimited در نظر بگیرید ، به اندازه کل دیسک فایل رشد خواهد کرد .

نکته : در صورتی که برای ذخیره سازی ، فایل های پایگاه داده ، مسیر داده شده موجود نباشد ، آن را ایجاد نخواهد کرد.

- افزودن یک فایل mdf دیگر به پایگاه داده :

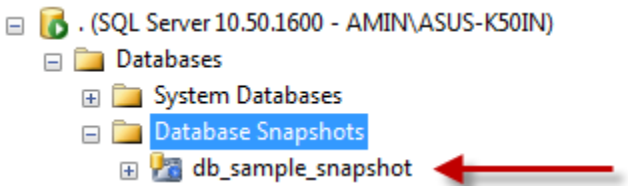
```
use db_sample
go
alter database db_sample
add
file(name='db_sample_mdf2',filename='D:\sql\db_sample_mdf2.mdf',size=10,maxsi
ze=150mb,filegrowth=5mb)
```

برای افزودن ابتدا باید پایگاه داده را ویرایش کرد ، پس از کلمه alter کمک می گیریم. حال می خواهیم فایل جدیدی اضافه نمائیم، پس از کلمه add file استفاده کرده و مانند قبل مشخصات فایل mdf جدید را وارد می کنیم.

- یک کپی فقط خواندنی از پایگاه داده :

```
use db_sample
go
create database db_sample_snapshot
ON(name='db_sample_mdf',filename='D:\Sql\db_sample_mdf_snapshot.mdf')
as snapshot of db_sample
go
```

پایگاه داده های snapshot را می توانید در مسیر Database->Database Snapshots مشاهده کنید.(تصویر زیر)



نکته : برای ایجاد پایگاه داده های snapshot ، درایو ها باید NTFS باشند.

- **الحاق کردن و جداکردن پایگاه داده**
  - **جداکردن پایگاه داده** : در صورتی که کپی فقط خواندنی (snapshot) از پایگاه داده داشته باشید ، نخواهید توانست پایگاه داده را جدا کنید. حال ما فرض می کنیم که snapshot را حذف نکرده ایم :

```
use master
go
exec sp_detach_db 'db_sample',true
```

سیستم پیغام زیر را نمایش خواهد داد و بیان می دارد که ابتدا باید فایل snapshot را حذف کرد :

```
Msg 3709, Level 16, State 1, Line 1
Cannot DETACH the database while the database snapshot "db_sample_snapshot" refers to it. Drop
that database first.
```

پس ابتدا باید فایل snapshot راحذف کرد پس از آن باید از تابع sp_detach_db استفاده نمود.

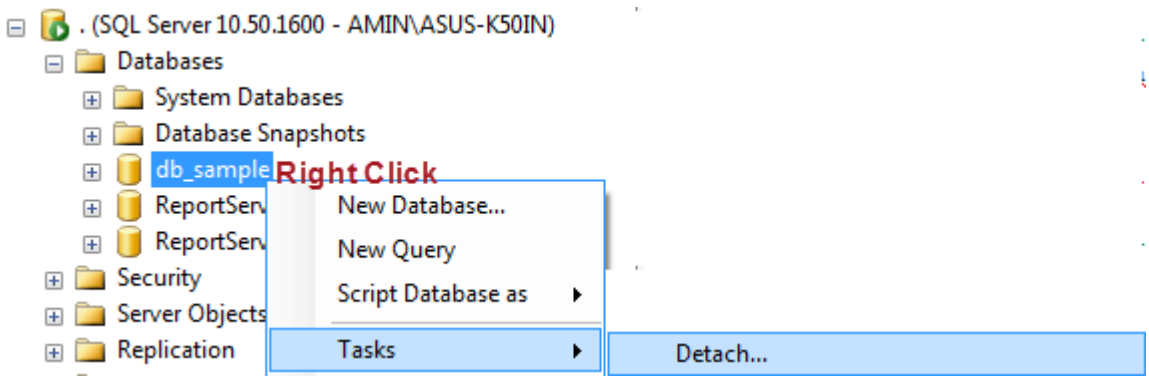
```
drop database db_sample_snapshot
go
use master
go
exec sp_detach_db 'db_sample',true
```

مثال بالا ، پایگاه داده db_sample را جدا کرده و همچنین Update Statistics را نادیده می گیرد.

```
drop database db_sample_snapshot
go
use master
go
exec sp_detach_db 'db_sample',false, @keepfulltextindexfile=true
```

در این مثال ، پایگاه داده را جدا کرده و شاخص Fullindex را ذخیره کرده و همچنین Update Statistics را هم درنظر می گیرد.

- **جداکردن یا detach کردن پایگاه داده به صورت ویزارد :**



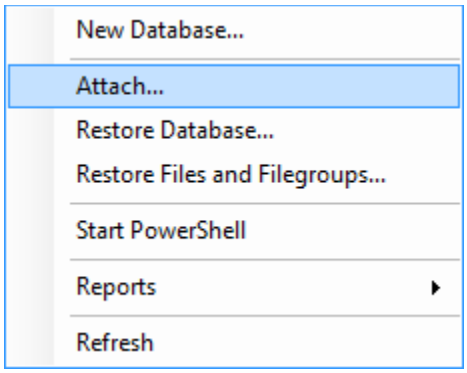
- **الحاق پایگاه داده :**

```
exec sp_attach_db
'db_sample','D:\sql\db_sample_mdf.mdf','D:\sql\db_sample_ldf.ldf'
```

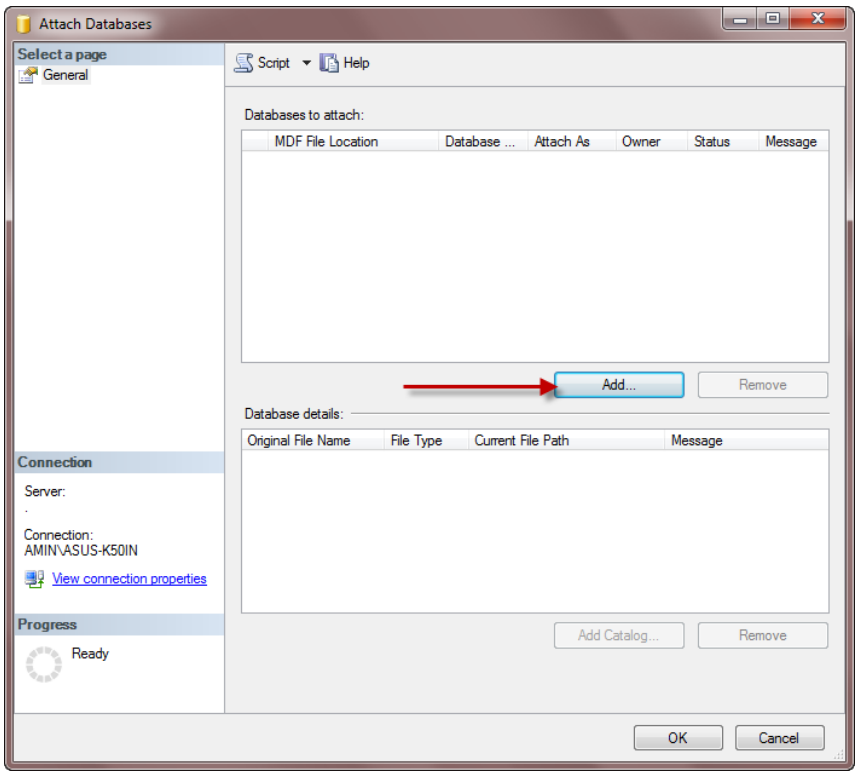
برای الحاق کردن پایگاه داده باید از تابع sp_attach_db استفاده نمود . پس از آن باید مسیر فایل mdf و مسیر فایل ldf را وارد کرد.

- **الحاق پایگاه داده از طریق ویزارد**

بر روی DataBase کلیک راست کرده و گزینه Attach را انتخاب نمائید.



پس از آن ، صفحه ای مانند زیر باز می شود و برای الحاق کردن پایگاه داده مورد نظر ، باید فایل های mdf و ldf پایگاه داده را از طریق دکمه Add ، به SSMS بیفزایید.



- **ایجاد جدول در پایگاه داده مورد نظر :**

می خواهیم جداولی که در فصل اول بیان شده را ایجاد نمائیم . (department - employee - project - work on)

- ایجاد جدول department :

```
use db2
go
create table Department
(
dept_no int not null primary key,
dept_name nvarchar(25) not null,
dept_location nvarchar(50) default 'tehran'
)
go
```

پس از ساخت جدول باید در بین دو پرانتز ، فیلد های جدول را با نوع آنها مشخص نمائید. برای هر فیلد می توانید محدودیت هایی را در نظر بگیرید :

- Null یا Not Null : همان طور که مشاهده می شود ، فیلد های dept_no و dept_name را با not null محدود کرده ایم . منظور از null این است که در صورت وارد کردن مقادیر در جدول ، بتوان آن فیلد را خالی رها کرد و مقداری را در آن ذخیره نکرد. not null برعکس null است و اجازه خالی رها کردن فیلد مورد نظر را نخواهد داد.

نکته : در صورتی که برای فیلدی ، محدودیت null یا not null را اعمال نکنید (مانند dept_location)، آن فیلد null درنظر گرفته می شود.

- **اگر برای بررسی تغییر و درج داده ها ، یک سری محدودیت هایی به کاربرده شود ، به آن ها محدودیت جامعیت گفته می شود .**
  - **محدودیت جامعیت یک سری مزایا دارد :**
    - افزایش قابلیت اعتماد داده ها : زمانی که شما محدودیت ها را در DBMS اعمال می کنید ، آن محدودیت ها یکبار نوشته می شوند ولی اگر بخواهید محدودیت ها با برنامه نویسی در برنامه کاربردی اعمال کنید ، اولا باید جایی که می خواهید تغییر یا درج انجام دهید ، محدودیت را اعمال کنید ، ثانیا ممکن است کاربر برنامه نویس ، اعمال محدودیت ها را فراموش کند که این باعث ناسازگاری داده ها می شود.

- کاهش زمان برنامه نویسی
- حفاظت آسان داده ها

- **دوگروه محدودیت جامعیت وجود دارد :**

- محدودیت جامعیت اعلانی : در سطح جدول و ستون می توانند باشند ، در صورتی که در سطح یک ستون باشد ، فقط بر روی آن ستون اعمال می شود ولی اگر در سطح جدول باشد ، ممکن است بر روی چند ستون اعمال شود.
- محدودیت جامعیت رویه ای

- **محدودیت های جامعیت اعلانی :**

- **Default** : در صورتی که بخواهید ، برای هر ستون یا فیلد مقداری پیش فرض قرار دهید ، از این محدودیت می توانید استفاده نمائید. در صورتی که هنگام افزودن مقادیر به جدول ، مقداری را در ستون مورد نظر وارد نکنید (مانند **dept_Location**) ، مقدا پیش فرضی (مانند **tehran**) برای آن در نظر گرفته می شود.

- **Unique** : گاهی اوقات ، ممکن است بخواهیم چند ستون جدول را که مقادیر یکتایی را دارند را به عنوان کلید درنظر بگیریم . (ستون هایی که شرایط کلید اصلی را داشته باشند را کلید کاندید می گویند). هر کلید کاندید در دستور **Create Table** یا **Alter Table** ، با عبارت **unique** تعریف می شود

- **Primary Key** : یک ستون یا گروهی از ستون هایی که مقدار منحصر به فردی را دارند را به عنوان کلید اصلی در نظر می گیریم .

- **Check** : اگر بخواهیم هنگام درج داده ها ، بررسی بر روی آن ها انجام شود ، **Check** استفاده می کنیم به مثال زیر توجه کنید : می خواهیم در ستونی فقط دو مقدار صفر یا یک درج شود و اگر عددی غیر از این دو بود، خطا نشان دهد :

```
use db2
go
create table Test
(
t1 int not null check(t1 in(0,1))
)
go
```

در جدول **test** ، ستونی تحت عنوان **t1** ساخته می شود که نوع آن **int** و نمی تواند خالی رها شود و همچنین باید دو عدد صفر یا یک در آن درج شود.

- **Foreign Key** : معمولا در پایگاه داده ، بین ستون های جداول ارتباط هایی وجود دارد که ارتباط را با کلید خارجی برقرار می کنند.

مشخصات کلید خارجی و کلید اصلی از لحاظ نوع و محدودیت ها و ... باید یکی باشد .

- **ساخت جدول کارمندان :**

```
use db2
go
create table employee
(
emp_no int not null primary key,
emp_fname nvarchar(13) not null,
emp_lname nvarchar(30) not null,
dept_no int not null foreign key(dept_no) References Department(dept_no)

)go
```

در این مثال ، **dept_no** کلید خارجی جدول **employee** می باشد. **foreign key** بیان کنند این است که فیلد **dept_no** ، کلید خارجی است . اما این کلید خارجی با ستون چه جدول یا جداولی مرتبط است ؟ برای بیان آن از **References** استفاده می شود که پس از آن نام جدول و در داخل پرانتز نام ستونی که با کلید خارجی مرتبط است ، نوشته می شود.

نکته : به جدولی که شامل کلید خارجی است ، جدول ارجاع کننده یا فرزند و به جدولی که شامل کلید اصلی متناظر است ، جدول پدر یا ارجاع شده گفته می شود.

نکته : حداکثر ۶۳ کلید خارجی در یک جدول می توان تعریف کرد.

نکته : اگر کلید اصلی در یک جدول بیش از یک ستون باشد ، نمی شود محدودیت ها را در سطح ستون اعمال کرد بلکه باید محدودیت ها را در سطح جدول اعمال کنیم .

- **ساخت جدول کارمندان با اعمال محدودیت در سطح جدول :**

```
create table employee
(
emp_no int not null ,
emp_fname nvarchar(13) not null,
emp_lname nvarchar(30) not null,
dept_no int not null ,
Constraint PK_emp_no primary key(emp_no),
Constraint FK_dept_no Foreign Key(dept_no) References Department(dept_no)

)
```

در این روش از کلمه Constraint استفاده می شود . برای هر Constraint باید یک نام متناظر با کاری که انجام می دهد ، درنظر گرفت (مثلا PK_emp_no ، که PK مخفف Primary Key می باشد.)

- **ساخت جدول پروژه ها :**

```
create table project
(
prj_no int not null,
prj_name nvarchar(50) not null,
budget money not null,
Constraint PK_prj_no Primary key(prj_no)
)
```

کلید اصلی در این جدول prj_no است .

- ساخت جدول کار کارمندان بر روی پروژه ها :

```
create table Work_On
(
emp_no int not null,
prj_no int not null,
job nvarchar(30) not null,
[date] date not null,
constraint FK_emp_no foreign key(emp_no) references employee(emp_no),
constraint FK_prj_no foreign key(prj_no) references project(prj_no),

)
go
```

- **گزینه های On Delete و On Update :**

اگر می خواهید تا هنگام حذف و تغییر مقادیر کلید اصلی ، Database Engine رفتاری متناسب داشته باشد می شود در هنگام حذف یک مقدار یا به روز رسانی ، رفتار های زیر را اعمال کرد :

- No Action : وقتی مقادیری از جدول پدر حذف یا تغییر شد ، تاثیری روی مقادیر فرزاندان نداشته باشد.
- Cascade : وقتی مقادیری از جدول پدر حذف یا تغییر شد روی مقادیر فرزاندان تاثیرگذار است.
- Set Null : وقتی مقادیری از جدول پدر حذف یا تغییر شد ، مقادیر فرزندان برابر با Null می شود.
- Set Default : وقتی مقادیری از جدول پدر حذف یا تغییر شد ، مقادیر فرزندان برابر با مقدار پیش فرض شان خواهد شد.

- **تعریف نوع داده های مستعار :**

برای ایجاد یک نوع داده مستعار در Sql از ساختار زیر استفاده می شود :

```
Create Type Mychar from Char(5)
go
```

نوع داده ای جدید با نام Mychar از نوع داده پایه Char ایجاد می کند.

- **تغییر پایگاه داده :**
  - تغییر نام پایگاه داده
  - اضافه یا حذف چند فایل به یا از پایگاه داده
  - اضافه یا حذف چند فایل تراکنش به یا از پایگاه داده
  - اضافه یا حذف فایل گروه
  - تغییر خصوصیات فایل گروه

- **اضافه و حذف کردن فایل های تراکنش :**

در صفحه ۱۸ به بحث ایجاد یک فایل mdf پرداخت شد . حال می خواهیم فایل ایجاد شده را حذف کرد :

```
use db_Sample
go
alter database db_Sample
Remove file db_sample_mdf2
go
```

- ایجاد یک Log File :

```
use db_Sample
go
alter database db_Sample
add log file
(name='db_sample_ldf2',filename='D:\sql\db_sample_ldf2.ldf',size=10,maxsize=1
50mb,filegrowth=5mb)
go
```

- **ایجاد File Group :**

ایجاد یک فایل گروه با نام fg1 در پایگاه داده :

```
use db_Sample
go
alter database db_Sample
add FileGROUP fg1
go
```

برای حذف فایل گروه کافی است به صورت زیر عمل شود :

```
use db_Sample
go
alter database db_Sample
Remove FileGROUP fg1
go
```

- تنظیم FileGroup به صورت فقط خواندنی :

قبل از بیان این موضوع ، اگر فایل گروه فاقد فایل باشد ، نمی توان تنظیماتی بر روی آن انجام داد ، برای همین ما فایلی را به فایل گروه اختصاص می دهیم .

```
use db_Sample
go
alter database db_Sample
add file
(name='db_sample_mdf3',filename='D:\sql\db_sample_mdf3.mdf',size=10,maxsize=1
50mb,filegrowth=5mb)
to filegroup fg1
go
```

همانطور که ملاحظه می شود ، to filegroup fg1 ، عمل انتقال فایل را به فایل گروه fg1 انجام می دهد . حال تنظیمات فایل گروه به صورت فقط خواندنی :

```
use db_Sample
go
alter database db_Sample
Modify  FileGROUP fg1  ReadOnly
go
```

برای تنظیم به صورت خواندن و نوشتن ، کافی است به جای Readonly از ReadWrite استفاده نمائید.

- **تنظیم فایل گروه به صورت فایل گروه پیش فرض :**

```
use db_Sample
go
alter database db_Sample
Modify  FileGROUP fg1  Default
go
```

- **تنظیمات پایگاه داده :**
  - **تنظیم پایگاه داده به صورت چند کاربره :**

```
use db_Sample
go
alter database db_Sample
set Multi_user
go
```

پایگاه داده را می توانید از لحاظ دسترسی به صورت تک کاربره نیز تنظیم نمائید ، برای این کار به جای Multi_User از Single_user استفاده نمائید.

  - **تنظیم پایگاه داده به صورت فقط خواندنی :**

```
use db_Sample
go
alter database db_Sample
set Read_only
```

```
go
```

حال اگر بخواهید ، پایگاه داده را به صورت خواندنی و نوشتی تنظیم کنید ، کافی است از Read_write استفاده کنید.

- **غیرفعال کردن پایگاه داده :**

```
use db_Sample
go
alter database db_Sample
set Offline
go
```

پس از این کار ، پایگاه داده غیرفعال شده و بسته می شود و پایگاه داده جاری به پایگاه داده master سوئیچ می شود.

- **فعال کردن پایگاه داده :**

```
alter database db_Sample
set Online
go
```

- **تغییر جداول :**

برای تغییر یک جدول کافی است از دستور Alter Table استفاده نمائید. با این کار می توان :

- اضافه یا حذف ستون از جدول
- تغییر خصوصیات جدول
- اضافه یا حذف محدودیت جامعیت
- فعال یا غیر فعال کردن محدودیت جامعیت
- تغییر نام جدول

```
use db_Sample
go
alter table employee
add tel_no char(11) not null
go
```

به جدول employee ، یک ستون با نام tel_no می افزاید.

```
use db_Sample
go
alter table employee
drop column tel_no
```

```
go
```

ستون tel_no را از جدول employee حذف می کند.

- **افزودن محدودیت به جدول :**

به صورت آزمایشی جدولی با نام test می سازیم .

```
use db2
go
create table test
(
      test_id int not null,
      test_name char(10) null,
)
go
```

حال می خواهیم به جدول بالا Constraint اضافه نمائیم :

```
use db2
go
alter table test
add constraint PK_test_id primary key(test_id)
go
```

همانطور که ملاحظه می شود برای افزودن Constraint باید از Add Constraint استفاده کرد. برای حذف Constraint کافی است از Drop Constraint استفاده نمائید :

```
use db2
go
alter table test
drop constraint PK_test_id
go
```

- **تغییر نام اشیای پایگاه داده با استفاده از sp_rename :**

```
use db_sample
exec sp_rename employee,employeee
go
```

تغییر نام از employee به employeee در پایگاه داده db_sample

نکته : سعی کنید از این تابع استفاده نکنید.چون ممکن است روی اشیای دیگر مثل رویه های ذخیره شده و ... تاثیر بگذارد.

فصل چهارم – پرس و جو ها و تغییر محتوای جداول

- **نمایش داده های جداول پایگاه داده :**

```
USE db_sample
GO
SELECT * FROM Department
```

برای نمایش داده های یک جدول ، می توان از دستور SELECT ، استفاده کرد.

- نکاتی در ارتباط با نمایش داده ها :
  - سمبل ستاره (*) در دستور **select** ، تمام فیلد های جدول را نمایش می دهد.
  - اگر بخواهیم برای یک ستون ، درهنگام نمایش آن ، نام مستعاری انتخاب نمائیم کافی است بعد از نام ستون از عبارت **as new_name** استفاده نمائید :

```
USE db_sample
GO
SELECT dept_no as Department_Number from department
```

هنوز ما داده ای به جداول اضافه نکرده ایم ، پس ابتدا روش های درج و به روزرسانی را توضیح خواهیم داد و سپس به روش های پرس وجو می پردازیم

- **درج در جداول با استفاده از Insert :**

```
USE db_sample
GO
INSERT INTO tbl_Department(dept_no,dept_name,dept_Location)
VALUES(1,'COMPUTER','BIRJAND')
GO
```

پس نام جدول می توانید نام ستون های جدول را درج کنید ، توجه داشته باشید اگر برای مثال بخواهید در یک ستون از جدول ، داده اضافه نمائید ، در بخش VALUES ، هم یک مقدار وارد نمائید .

نکته : در صورتی که نام ستون های جدول را بعد از نام جدول اضافه نکنید ، تمام ستون های جدول در نظر گرفته می شود .

```
USE db_sample
GO
INSERT INTO tbl_Department
VALUES(1,'ACCOUNTING','BIRJAND')
GO
```

با وارد کردن این دستور ، با خطا مواجه خواهید شد ، چون مقدار dept_no ، به صورت کلید اصلی در نظر گرفته شده و باید منحصر به فرد باشد.

```
USE db_sample
GO
INSERT INTO tbl_Department
VALUES(2,'ACCOUNTING','BIRJAND')
GO
```

در مثال زیر ، فقط می خواهیم در دو ستون عمل درج را انجام دهیم ، در صورتی که اگر در ستون سوم مقداری درج نکنید ، با خطا مواجه می شوید (چون not null) اما برای آن Default هم درنظر شده که آن هم tehran است.

```
USE db_sample
GO
INSERT INTO tbl_Department(dept_no,dept_name)
VALUES(3,'ACCOUNTING')
GO
```

پس بعد از درج اگر دستور ، پرس و جو از جدول را وارد نمائید ، متوجه می شوید که مقدار پیش فرض درج شده است.

```
SELECT * FROM tbl_Department
```

	dept_no	dept_name	dept_Location
1	1	COMPUTER	BIRJAND
2	2	ACCOUNTING	BIRJAND
3	3	ACCOUNTING	Tehran

- **نمایش داده های مورد نیاز :**

اگر بخواهیم از جدول tbl_depratment ، سطر هایی را بازیابی کنیم ، که محل های آن در شهر Birjand باشد می توان به صورت زیر از where استفاده کنیم :

```
SELECT * FROM tbl_Department
WHERE dept_Location='BIRJAND'
```

dept_no	dept_name	dept_Location
1	COMPUTER	BIRJAND
2	ACCOUNTING	BIRJAND

در مقابل عبارت where می توان از علامت های زیر نیز استفاده کرد :

>
<
>=
<=
!=
!>
!<

مثال : نمایش داده هایی که مقدار dept_no شان بزرگتر مساوی عدد دو است :

```
SELECT * FROM tbl_Department
WHERE dept_no>=2
```

dept_no	dept_name	dept_Location
2	ACCOUNTING	BIRJAND
3	ACCOUNTING	Tehran

توجه داشته باشید که شرط مقابل عبارت where می تواند چند شرطی باشد ، برای مثال می خواهیم رکوردهایی را نمایش دهیم که محل آن ها Birjand و dept_name آنها ACCOUNTING باشد :

```
SELECT * FROM tbl_Department
WHERE dept_name='ACCOUNTING' and dept_Location='BIRJAND'
```

dept_no	dept_name	dept_Location
2	ACCOUNTING	BIRJAND

همانطور که ملاحظه می شود ، عبارات چند شرطی را باید با عملگرهای and ، or و Not از هم جدا نمائید. (اولویت عملگر not از همه بالاتر است و سپس and و پس از آن or)

مثال : می خواهیم شهرهایی را بازیابی کنیم که محل شان بیرجند است :

```
SELECT dept_Location from tbl_Department
WHERE dept_Location='BIRJAND'
```

dept_Location
BIRJAND
BIRJAND

مشکلی که در این جا با آن مواجه هستیم ، این است که نام شهر دوبار تکرار شده است ، برای حل این مشکل می توان از عبارت Distinct قبل از نام ستون استفاده کرد :

حل مشکل :

```
SELECT distinct dept_Location from tbl_Department
WHERE dept_Location='BIRJAND'
```

dept_Location
BIRJAND

- **آشنایی با عملگر های In و Between :**

عملگر **In** برابر با چند شرط **Or** است ، برای مثال می خواهیم رکوردهایی را نمایش دهیم که **dept_no** شان ، یا یک و یا سه باشد :

```
SELECT * FROM tbl_Department
WHERE dept_no In(1,3)
```

dept_no	dept_name	dept_Location
1	COMPUTER	BIRJAND
3	ACCOUNTING	Tehran

همچنین می توان عملگر **In** را با عملگر **Not** هم به کار برد ، برای مثال می خواهیم رکوردهایی را نمایش دهیم که شامل **dept_no** ، یک و سه نباشد :

```
SELECT * FROM tbl_Department
WHERE dept_no not In(1,3)
```

dept_no	dept_name	dept_Location
2	ACCOUNTING	BIRJAND

اگر بخواهیم مقادیر بین دو مقدار را برگردانیم ، می توان از عملگر **Between** ، استفاده نمود. عکس عملگر **between** ، **not between** است

مثالی از **not between** : نمایش رکوردهایی که بین یک و سه نیست

```
SELECT * FROM tbl_Department
WHERE dept_no not between 2 and 3
```

dept_no	dept_name	dept_Location
1	COMPUTER	BIRJAND

مثال : درج چند رکورد در جدول کارمندان :

```
INSERT INTO tbl_Employee
VALUES
('Hasan','Hasani',1),
('Ali','Alavi',1),
('Amin','Amini',2)
```

مثال : درج چند رکورد در جدول پروژه ها :

```
INSERT INTO tbl_Project
VALUES
(N'مرکزی ساختمان',100000),
(N'موشک ساخت',200000),
(N'ای رسانه چند افزارهای نرم ساخت',1500000)
```

توجه داشته باشید برای درج عبارات فارسی ، باید قبل از عبارت کلمه N را نیز بکار ببرید ، در غیر این صورت داده ها به صورت علامت سوال نمایش داده خواهند شد.

مثال : درج رکورد در جدول Work_on :

```
INSERT INTO Work_on
VALUES
(100,4,N'ناظر',GETDATE())
```

مثال : نام و نام خانوادگی کسی که بر روی پروژه نظارت بر ساخت موشک فعالیت دارد :

```
SELECT emp_fname,emp_lname FROM tbl_Employee
WHERE emp_no = (SELECT emp_no FROM work_on
WHERE prj_no = (SELECT Prj_no from tbl_Project
WHERE Prj_name=N'موشک ساخت'))
```

emp_fname	emp_lname
Hasan	Hasani

ابتدا باید شماره پروژه ساخت موشک را به دست بیاوریم ، سپس از طریق آن باید شماره کارمند را به دست آوریم و بعد از آن می توان نام و نام خانوادگی را به دست آورد.

- **آشنایی با دستور Like :**

می خواهیم از جدول کارمندان ، نام کسانی را به دست آوریم که ابتدای نام شان برابر با **A** باشد :

```
SELECT * FROM tbl_Employee
WHERE emp_fname LIKE 'A%'
```

emp_fname	emp_lname	dept_no	emp_no
Ali	Alavi	1	101
Amin	Amini	2	102

مثال : نمایش نام کسانی از جدول کارمندان که آخر نام شان با حرف **n** تمام می شود :

```
SELECT * FROM tbl_Employee
WHERE emp_fname LIKE '%n'
```

emp_fname	emp_lname	dept_no	emp_no
Hasan	Hasani	1	100
Amin	Amini	2	102

- **مرتب کردن رکورد ها (نزولی / صعودی)**

برای مرتب کردن کافی است بعد از دستور **Select** ، از **Order By** استفاده نمائید :

```
SELECT * FROM tbl_Employee
ORDER BY emp_fname
```

emp_fname	emp_lname	dept_no	emp_no
Ali	Alavi	1	101
Amin	Amini	2	102
Hasan	Hasani	1	100

در صورتی که نزولی یا صعودی را معین نکنید ، به طور پیش فرض صعودی است . اما برای نزولی کردن کافی است از **desc** بعد از **order by** استفاده کنید :

```
SELECT * FROM tbl_Employee
ORDER BY emp_fname desc
```

emp_fname	emp_lname	dept_no	emp_no
Hasan	Hasani	1	100
Amin	Amini	2	102
Ali	Alavi	1	101

می خواهیم متوجه شویم که در جدول employee ، آیا ستونی وجود دارد که Identity داشته باشد و اگر دارد مقدار آن و مقدار افزایشی آن چیست ؟

```
SELECT IDENT_SEED('tbl_employee') as SEED,IDENT_INCR('tbl_employee') as
INCREMENT
```

نکته : در یک جدول یکی از فیلدهای آن دارای خصوصیت Identity است و اگر بخواهیم این خصوصیت را غیرفعال کنیم باید دستور زیر را وارد نمائید : (در این مثال ، خصوصیت فیلد جدول TBL_EMPLOYEE را غیر فعال و فعال می کنیم).

```
SET IDENTITY_INSERT TBL_EMPLOYEE OFF
```

برای فعال کردن هم :

```
SET IDENTITY_INSERT TBL_EMPLOYEE ON
```

- **آشنایی با دستورات اجتماع ، اشتراک و تفاضل**
  - اجتماع : می خواهیم از دو پرس و جو زیر اجتماع بگیریم ، توجه داشته باشید که در اجتماع ، اشتراک ، تفاضل فیلدی مشترک باید وجود داشته باشد.

```
SELECT emp_no FROM tbl_Employee
WHERE dept_no=1
```

emp_no
100
101

```
SELECT emp_no FROM Work_on
WHERE prj_no=4
```

emp_no
100

برای اجتماع این دو جدول باید از union استفاده نمود :

```
SELECT emp_no FROM tbl_Employee
WHERE dept_no=1
UNION
SELECT emp_no FROM Work_on
WHERE prj_no=4
```

اشتراک :

```
SELECT emp_no FROM tbl_Employee
WHERE dept_no=1
INTERSECT
SELECT emp_no FROM Work_on
WHERE prj_no=4
```

emp_no
100

اجتماع :

```
SELECT emp_no FROM tbl_Employee
WHERE dept_no=1
EXCEPT
SELECT emp_no FROM Work_on
WHERE prj_no=4
```

emp_no
101

- **جدول موقت**

جدول موقت یکی از اشیای پایگاه داده است که توسط سیستم پایگاه داده ذخیره می شود ، آن ها در پایگاه داده سیستمی **tempdb** ذخیره می شوند و پیشوند آن ها **#** است . پس از پایان جلسه کاری ، جداول موقت به طور خودکار حذف می شوند .

ساخت جدول موقت :

```
CREATE TABLE #TBL_PROJECT
(
	prj_no int not null,
	prj_name nvarchar(30) not null
)
```

ساخت جدول موقت از روی جدول دیگر و همراه با داده های آن :

```
SELECT prj_no,prj_name
INTO
#TBL_PROJECT
FROM tbl_Project
```

- **پیوند**
  - پیوند طبیعی (inner join)
  - ضرب دکارتی (Cross join)

- **پیوند طبیعی :**

```
SELECT tbl_Employee.*,tbl_Department.*
FROM tbl_Employee INNER JOIN tbl_Department
ON tbl_Employee.dept_no=tbl_Department.dept_no
```

همه مقادیر جدول کارمندان و همه مقادیر جدول ادارات براساس dept_no با هم پیوند می خورند .

	emp_fname	emp_lname	dept_no	emp_no	dept_no	dept_name	dept_Location
1	Hasan	Hasani	1	100	1	COMPUTER	BIRJAND
2	Ali	Alavi	1	101	1	COMPUTER	BIRJAND
3	Amin	Amini	2	102	2	ACCOUNTING	BIRJAND

- **ضرب دکارتی :**

می خواهیم دو جدول را با هم پیوند دهیم اما نه براساس فیلدی خاص برای این کار از ضرب دکارتی استفاده می شود :

```
SELECT
tbl_Employee.emp_fname,emp_lname,tbl_Department.dept_name,dept_Location
FROM tbl_Employee CROSS JOIN tbl_Department
```

- **به روز رسانی مقادیر جداول**

در جدول کارمندان ، به اشتباه ، نام آقای امیر امینی ، امین امینی درج شده است ، برای ویرایش آن باید از دستور UPDATE استفاده کنیم :

emp_fname	emp_lname	dept_no	emp_no
Hasan	Hasani	1	100
Ali	Alavi	1	101
Amin	Amini	2	102

```
UPDATE tbl_Employee
SET emp_fname='Amir'
WHERE emp_no=102
```

- **حذف رکورد**

برای حذف رکورد های یک جدول از دستور Delete استفاده می شود ، در صورتی که بخواهید تمام رکوردهای جدولی را پاک کنید کافی است :

```
DELETE TBL_PROJECT
```

البته می توان در حذف رکورد ها نیز شرطی را در نظر گرفت . حال اگر بخواهیم تمام رکوردهای یک جدول را حذف کنیم ، دستور سریع تری نیز وجود دارد و آن Truncate table نام دارد . علت سریع بودن آن ؛ این است که صفحه به صفحه رکوردها را پاک می کند.

- **آشنایی با دستور output :**

در صورتی که بخواهیم ، رکوردهای حذف شده ، درج شده و یا به روز رسانی شده را در پس از اعمال دستورات یاد شده نمایش دهیم از output استفاده می کنیم :

```
DECLARE @DEL TABLE(PRJ_NO INT,PRJ_NAME NVARCHAR(50));
DELETE TBL_PROJECT
OUTPUT DELETED.PRJ_NO,DELETED.PRJ_NAME INTO @DEL
SELECT * FROM @DEL
```

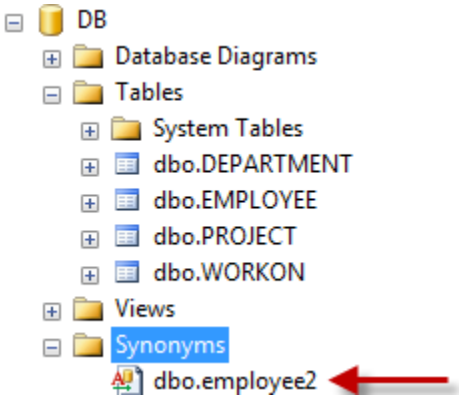
ابتدا یک متغیر جدولی با نام @DEL ایجاد می کنیم و به آن فیلدهای متناظر را می دهیم . بعد از حذف جدول می توان از دستور Output Deleted.prj_no,Deleted.prj_name into @del استفاده نمود تا رکوردهای حذف شده را در متغیر بریزد .

- **ساخت یک شی از پایگاه داده با نامی دیگر:**

برای مثال می خواهیم از جدول کارمندان یک کپی با نامی دیگر داشته باشیم برای این کار از Create Synonym استفاده می کنیم :

```
create synonym employee2
for employee
```

این جدول در پوشه Synonyms ذخیره می شود.



- **آشنایی با دستور Merge:**

فرض کنید دو جدول مانند هم داریم که می خواهیم مقادیر دو جدول را مانند هم کنیم .

```
USE DB
GO
CREATE TABLE PROJECT2
(
	PRJ_NO INT NOT NULL,
	PRJ_NAME NVARCHAR(20) NOT NULL,
)
GO

MERGE INTO PROJECT2 A
USING (SELECT PRJ_NO,PRJ_NAME FROM PROJECT) B
ON(A.PRJ_NO=B.PRJ_NO)
WHEN MATCHED THEN
	UPDATE SET A.PRJ_NAME=B.PRJ_NAME
WHEN NOT MATCHED THEN
INSERT(PRJ_NO,PRJ_NAME) VALUES (B.PRJ_NO,B.PRJ_NAME);
GO
```

ابتدا یک جدول جدید با نام Project2 ساختیم ، حال می خواهیم این جدول را با جدول Project مقایسه کرده و اگر کلیدها برابر بود ، مقادیر موجود به روز رسانی شود واگر کلید ها برابر نبودند ، مقادیر جدیدی به جدول افزوده شود.

- **تمرین**

۱- جزئیات کامل همه ادارات را به دست آورید .

```
USE DB
GO
SELECT * FROM DEPARTMENT
```

۲- اسامی و شماره همه ادارات قرار گرفته در شهر TEHRAN را به دست آورید.

```
USE DB
GO
SELECT DEPT_NO,DEPT_NAME FROM DEPARTMENT
WHERE LOCATION='TEHRAN'
```

۳- نام و نام خانوادگی همه کارمندانی که شماره کارمندی آن ها بزرگتر مساوری ۲ هستند را بیابید.

```
USE DB
GO
SELECT EMP_FNAME,EMP_LNAME FROM EMPLOYEE
WHERE EMP_NO>=2
```

۴- نام پروژه هایی را که بودجه آن ها بیش تر از 1000000 تومان است را به دست آورید.

```
USE DB
GO
SELECT PRJ_NAME FROM PROJECT
WHERE BUDGET>1000000.00
```

۵- شماره کارمندانی را بیابید که در پروژه 1 و 101 یا هر دو کار می کنند.

```
USE DB
GO
SELECT EMP_NO FROM WORKON
WHERE PRJ_NO=1 OR PRJ_NO=101
```

۶- شماره کارمندی و اسامی کارمندانی را بیابید که به اداره ۱ تعلق ندارند.

```
USE DB
GO
SELECT EMP_NO,EMP_FNAME,EMP_LNAME FROM EMPLOYEE
```

```
WHERE DEPT_NO!=2
```

۷- همه اطلاعات کارمندانی را بیابید که شماره کارمندی آنها ۱۰۰ و ۱۰۲ یا ۱۰۴ باشد .

```
USE DB
GO
SELECT EMP_NO,EMP_FNAME,EMP_LNAME FROM EMPLOYEE
WHERE EMP_NO IN(100,102,104)
```

۸- همه اطلاعات کارمندانی را بیابید که شماره کارمندی آنها ۱۰۰ و ۱۰۲ یا ۱۰۴ نباشد .

```
USE DB
GO
SELECT EMP_NO,EMP_FNAME,EMP_LNAME FROM EMPLOYEE
WHERE EMP_NO  NOT IN(100,102,104)
```

۹- اسامی و بودجه پروژه هایی را بیابید که بودجه آنها بین 1500000 و30000000 تومان باشد.

```
USE DB
GO
SELECT PRJ_NAME,BUDGET FROM PROJECT
WHERE BUDGET BETWEEN 15000000 AND 30000000
```

۱۰- شماره کارمندان و پروژه های متناظر آن هایی را بیابید که روی پروژه 1 کار می کنند و کار مشخصی ندارند.

```
USE DB
GO
SELECT EMP_NO,PRJ_NO FROM WORKON
WHERE PRJ_NO=1 AND JOB IS NULL
```

۱۱- نام و نام خانوادگی کارمندانی را به دست آورید که در اداره ACCOUNTING کار می کنند.

```
USE DB
GO
(SELECT * FROM EMPLOYEE
WHERE DEPT_NO =
(SELECT DEPT_NO FROM DEPARTMENT
WHERE DEPT_NAME='ACCOUNTING'))
```

۱۲-جزئیات کارمندانی را به دست آورید که اداره آن ها در BIRJAND قرار دارد.

```
USE DB
GO
(SELECT EMP_FNAME,EMP_LNAME FROM EMPLOYEE
WHERE DEPT_NO IN
(SELECT DEPT_NO FROM DEPARTMENT
WHERE LOCATION='BIRJAND'))
```

۱۳- نام خانوادگی کارمندانی را به دست آورید که روی پروژه ساخت موشک فعالیت دارند.

```
USE DB
GO
SELECT EMP_LNAME FROM EMPLOYEE
WHERE EMP_NO IN
(SELECT EMP_NO FROM WORKON
WHERE PRJ_NO IN
(SELECT PRJ_NO FROM PROJECT
WHERE PRJ_NAME=N'ساخت ماکت'))
```

۱۴- همه شغل های کارمندان را به دست آورید .

```
SELECT JOB FROM WORKON
 GROUP BY JOB
```

۱۵-کوچکترین شماره کارمندی را به دست آورید .

```
SELECT MIN(EMP_NO)AS MIN_EMP FROM EMPLOYEE
```

۱۶-شماره کارمندی را بیابید که دیرتر از بقیه به جدول WORKON وارد شده است .

```
SELECT EMP_NO FROM WORKON
 WHERE [DATE] IN(
 SELECT MAX([DATE]) FROM WORKON)
```

۱۷-مجموع بودجه همه پروژه ها را محاسبه کنید.

```
SELECT SUM(BUDGET) AS BUDGET FROM PROJECT
```

۱۸- میانگین همه بودجه های بیشتر از 2000000 را محاسبه کنید.

```
SELECT AVG(BUDGET) FROM PROJECT
 WHERE BUDGET>2000000
```

۱۹- همه شغل های متفاوت هر پروژه را بشمارید .

```
SELECT PRJ_NO,COUNT(JOB) AS JOB_COUNT FROM WORKON
GROUP BY PRJ_NO
```

۲۰- تعداد شغل های هر پروژه را به دست آورید .

```
SELECT JOB,COUNT(*)JOB_COUNT FROM WORKON
GROUP BY JOB
```

۲۱- بررسی این که بودجه پروژه ای اگر کمتر از 1500000 باشد در ۲۰ درصد ضرب ، اگر بین 1500000 و 5000000 باشد در ۵ درصد ضرب و در غیر این صورت در ۳ درصد ضرب شود.

```
UPDATE PROJECT
SET BUDGET=
CASE
WHEN BUDGET>=1500000 THEN BUDGET*(20/100)
WHEN BUDGET BETWEEN 1500000 AND 5000000 THEN BUDGET*(5/100)
ELSE BUDGET*(3/100)
END
FROM PROJECT;
```

۲۲- خصوصیات کارمندان و اداراتی که در آن کار می کند را نشان دهید .

```
SELECT EMPLOYEE.* ,DEPARTMENT.*
FROM EMPLOYEE INNER JOIN DEPARTMENT
ON EMPLOYEE.DEPT_NO=DEPARTMENT.DEPT_NO
```

۲۳- جزئیات کارمندانی را بیابید که روی پروژه ساخت ماکت کار می کنند.

```
SELECT * FROM EMPLOYEE
WHERE EMP_NO IN(
SELECT EMP_NO FROM WORKON
WHERE PRJ_NO IN
(SELECT PRJ_NO FROM PROJECT
WHERE PRJ_NAME=N'ماکت ساخت'))
```

```
SELECT EMP_FNAME,EMP_LNAME,DEPT_NO FROM PROJECT JOIN  WORKON
ON WORKON.PRJ_NO=PROJECT.PRJ_NO JOIN EMPLOYEE
ON WORKON.EMP_NO=EMPLOYEE.EMP_NO
WHERE PRJ_NAME=N'ماکت ساخت'
```

۲۴- برای جدول Department یک نام متسعار در نظر بگیرید.

```
CREATE SYNONYM DEP FOR DEPARTMENT
```

فصل چهارم – رویه های ذخیره شده و توابع کاربری

در این فصل بسته ها و روتین ها معرفی می شوند. بسته حاوی دنباله ای از دستورات T-SQL و بسط های رویه ای است . روتین هم می تواند یک رویه ذخیره شده یا یک تابع کاربری باشد .

مزیت بسته نسبت به گروهی از دستورات تکی در این است که اجرای لحظه ای همه دستورات می تواند مزایای بهره وری چشم گیری را فراهم کند.

- بلوک دستورات

برای ساختن واحدهایی با یک یا چند دستور T-SQL از بلوک استفاده می شود. هر بلوک با دستور begin شروع شده و با دستور end خاتمه می یابد . از طریق دستور IF در بلوک می توان چند دستور را براساس برقراری یا عدم برقراری شرط اجرا کنیم .

مثال : می خواهیم بررسی کنیم اگر تعداد رکوردهای جدول کارمندان برابر ۵ تا بود ، TRUE چاپ کرده و در غیر این صورت ؛ ابتدا FALSE چاپ کرده و سپس تمامی رکورهای جدول کارمندان را نمایش دهد.

```
IF((SELECT COUNT(EMP_NO) FROM EMPLOYEE )=5)
      PRINT 'TRUE'
ELSE
BEGIN
      PRINT 'FALSE'
      SELECT * FROM EMPLOYEE
END
```

مثال : می خواهیم بررسی کنیم اگر رکوردی با شماره کارمندی ۱۰۰ وجود ندارد ، رکورد جدیدی با نام MOSEHN KAMALI را درج کن در غیر این صورت فامیل این نفر را به KAMALI عوض کن.

```
IF NOT EXISTS(SELECT * FROM EMPLOYEE WHERE EMP_NO=100)
BEGIN
      INSERT INTO EMPLOYEE VALUES ('MOHSEN','KAMALI',4)
END
ELSE
BEGIN
      UPDATE EMPLOYEE SET EMP_LNAME='KAMALI' WHERE EMP_NO=100
END
```

مثال : اگر جمع بودجه جدول پروژه ها کمتر از 1000,000.00 باشد ، بودجه را در ۱.۱ ضرب کن و اگر جمع بودجه برابر صفر باشد ، بودجه را با 60,000.00 جمع کن و در آخر جدول پروژه ها را نشان بده.

```
WHILE (SELECT SUM(BUDGET) FROM PROJECT)<1000000.00
BEGIN
      UPDATE PROJECT SET BUDGET=BUDGET*1.1
      IF(SELECT SUM(BUDGET) FROM PROJECT)=0
      BEGIN
            UPDATE PROJECT SET BUDGET=BUDGET+60000
      END
      SELECT * FROM PROJECT
END
```

نکته : در دستور WHILE هم می توان از BREAK و CONTINUE نیز می توان استفاده نمود.

مثال : می خواهیم بررسی کنیم اگر در جدول ادرات ، نام ACCOUNTING در شهر BIRJAND موجود است ، پیغام ACCOUNTING EXISTS را نمایش داده و در غیر این صورت ، این مقادیر را درج کرده و پیغام ROW ADDED را نمایش دهد.

```
DECLARE @NAME NVARCHAR(20)='ACCOUNTING'
DECLARE @LOCATION NVARCHAR(20)='BIRJAND'
DECLARE @EXISTS BIT = 0
IF EXISTS ( SELECT * FROM DEPARTMENT WHERE DEPT_NAME=@NAME AND
LOCATION=@LOCATION )
BEGIN
      SET @EXISTS = 1
      GOTO SKIPINSERT
END
ELSE
BEGIN
      INSERT INTO DEPARTMENT VALUES (@NAME,@LOCATION)
END
SKIPINSERT:
IF @EXISTS = 1
BEGIN
      PRINT @NAME + 'ALREADY EXISTS'
END
ELSE
BEGIN
      PRINT 'ROW ADDED'
END
```

- **دستورات رویه ای گوناگون**
  - RETURN : همانند دستور BREAK در WHILE است یعنی اجرای دستورات جاری را متوقف کرده و دستورات بعد از حلقه را اجرا می کند.
  - GOTO : به یک برچسب در داخل بسته از دستورات پرش می کند.
  - RAISEERROR : یک پیام خطای کاربری تولید می کند و پرچم خطای سیستمی را نیز تنظیم می کند . شماره خطاهای کاربری باید بزرگتر از ۵۰۰۰۰ باشند . [مقادیر خطا در متغیر سراسری @@ERROR ذخیره می شوند.]
  - WAITFOR : یک فاصله زمانی (اگر گزینه DELAY تنظیم شود) یا یک زمان مشخص (اگر گزینه TIME تنظیم شود) تعریف می کند و سیستم را مجبور می کند به اندازه زمان مشخص شده تا قبل از اجرای دستور بعدی منتظر بماند.

WAITFOR {DELAY 'TIME'|TIME 'TIME'|TIMEOUT 'TIME'}

```
PRINT 'HELLO'
WAITFOR DELAY '00:00:10'
PRINT 'GOODBYE'
```

به مدت ۱۰ ثانیه صبر کرده و سپس پیغام GOODBYE را نمایش خواهد داد.

- **اداره کردن استثناها با دستورات TRY و CATCH**

هر مشکل که از ادامه برنامه به نحوی جلوگیری می کند را استثنا می گویند . با وجود چنین مشکلی نمی توان برنامه را ادامه دهیم . برای حل این مشکل ، مشکل موجود را به بخش دیگری از برنامه موکول می کنیم تا استثنا ها را مدیریت کند.

دستور TRY نقش گرفتن استثنا را ایفا می کند ، اگر استثنا در TRY رخ دهد ، استثنا به بخش دیگری موکول شده تا استثنا مدیریت شود ، این بخش CATCH نام دارد.

```
BEGIN TRY
BEGIN TRANSACTION
      INSERT INTO EMPLOYEE VALUES ('REZA','RASOULI',4);
      INSERT INTO EMPLOYEE VALUES('AMIN','KAMALI',1);
COMMIT TRANSACTION
PRINT 'TRANSACTION COMMITED'
```

```
END TRY
BEGIN CATCH
ROLLBACK
PRINT 'TRANSACTION ROLL BACK'
END CATCH
```

- **معرفی تعدادی از توابع خطا**

توابع زیر برای اداره کردن خطاها مفید می باشند :

- ERROR_LINE() : شماره خطی که در آن خطا رخ داده است را بر می گرداند.
- ERROR_NUMBER() : شماره آخرین خطای رخ داده را برمی گرداند.
- ERROR_MESSAGE() : پیام آخرین خطا را برمی گرداند.
- ERROR_PROCEDURE() : نام پروسه ای را برمی گرداند که خطا در آن رخ داده است.

نکته ای که در این جا باید یادآورد شد این است که این توابع را باید در CATCH به کار برد ، در غیر این صورت مقدار NULL برمی گردانند.

- **بررسی پیام های خطا در SQL SERVER**

SQL SERVER به کاربران خود این امکان را داده که پیام های خطای خود را به سیستم اضافه نمایند. رویه سیستمی SP_ADDMESSAGE برای اضافه کردن پیام های سفارشی به کار می رود :

```
EXEC SP_ADDMESSGAE 50001,10,'THIS IS A EXAMPLE','US_ENGLISH'
```

50001 : شماره پیام است و باید عددی بزرگتر از یک باشد.

10 : میزان شدت خطا را مشخص می کند.

THIS IS A EXAMPLE : در این قسمت می توانید متن مورد نظر خود را وارد نمائید.

US_ENGLISH : زبان به کار رفته در متن پیام را مشخص می کند.

- **رویه های ذخیره شده**

رویه ذخیره شده ، یک بسته خاصی است که با استفاده از زبان T-SQL و بسط های رویه ای نوشته می شود . تفاوت رویه با بسته در این است که رویه به عنوان شی پایگاه داده ذخیره می شود.

رویه های ذخیره شده ، قبل از ذخیره شدن ، کامپایل می شوند. **رویه ها کامپایل تکراری را حذف می کنند و بهره وری را افزایش می دهند و هم چنین حجم اطلاعات ارسالی و دریافتی پایگاه داده را کاهش می دهند.**

- **ایجاد و اجرای رویه های ذخیره شده**

رویه های ذخیره شده توسط دستور CREATE PROCUDURE ایجاد می شوند .

مثال : ایجاد رویه ، درج در جدول کارمندان :

```
CREATE PROCEDURE SP_INSERT_EMPLOYEE (
@EMP_FNAME NVARCHAR(13),
@EMP_LNAME NVARCHAR(25),
@DEPT_NO INT)
AS
INSERT INTO EMPLOYEE VALUES(@EMP_FNAME,@EMP_LNAME,@DEPT_NO)
```

برای اجرای رویه ایجاد شده :

```
EXEC SP_INSERT_EMPLOYEE 'REZA','GHOLAMI',4
```

برای حذف رویه های ذخیره شده :

```
DROP PROCEDURE SP_INSERT_EMPLOYEE
```

مثال : ایجاد یک رویه برای حذف مقدار

```
CREATE PROC SP_DEL_EMPLOYEE
(
@EMP_FNAME NVARCHAR(13),
@EMP_LNAME NVARCHAR(25),
@DEPT_NO INT
)
AS
```

```
DELETE EMPLOYEE WHERE EMP_FNAME=@EMP_FNAME AND EMP_LNAME=@EMP_LNAME AND
DEPT_NO=@DEPT_NO
```

- **تغییر ساختار رویه های ذخیره شده**

تغییر ساختار یک رویه همانند ساخت یک رویه می باشد اما به جای دستور CREATE PROCEDURE از دستور ALTER PROCEDURE استفاده می شود.

مثال : تغییر مثال قبل :

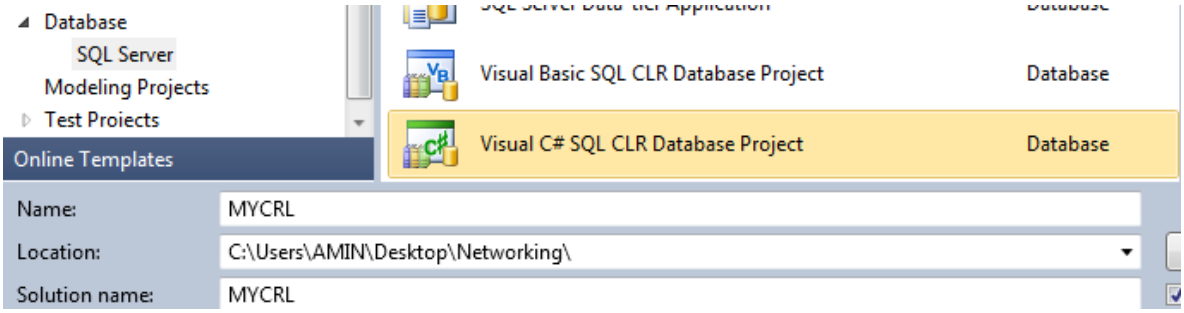
```
ALTER PROC SP_DEL_EMPLOYEE
(
@EMP_FNAME NVARCHAR(13),
@EMP_LNAME NVARCHAR(25)
)
AS
DELETE EMPLOYEE WHERE EMP_FNAME=@EMP_FNAME AND EMP_LNAME=@EMP_LNAME
```

- **CLR و رویه های ذخیره شده**

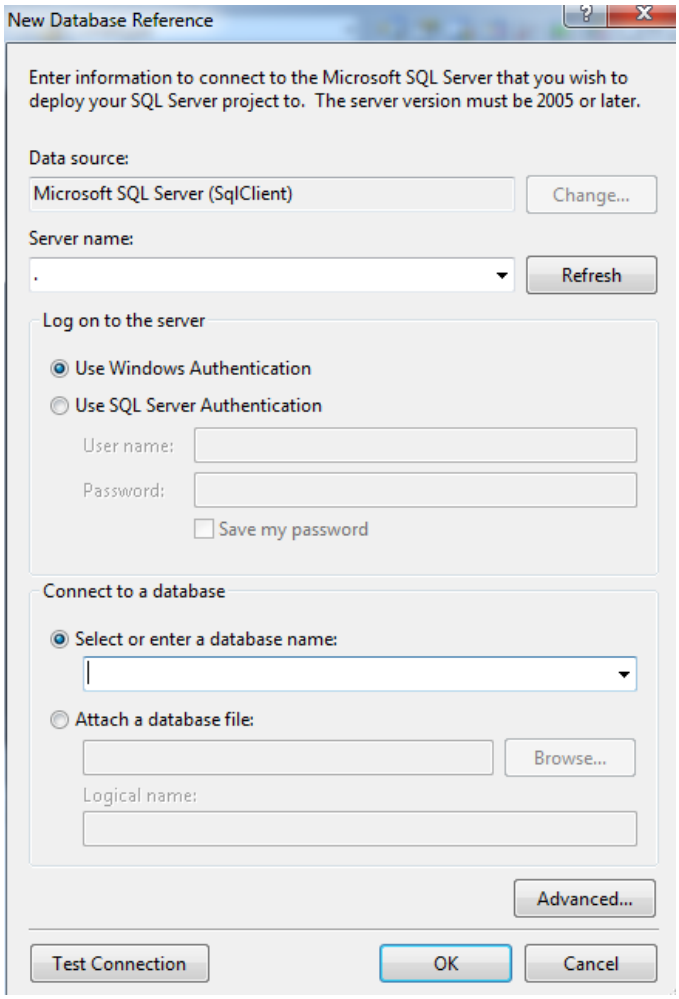
SQL SERVER این امکان را به ما می دهد که بتوان اشیای مختلف پایگاه داده را با استفاده از C# ، VB ایجاد کرد.

مراحل کار به صورت زیر است :

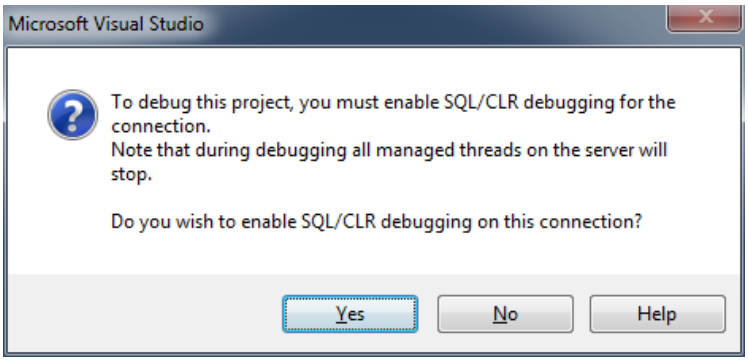
ابتدا در Visual Studio ، از منوی File ، گزینه New و سپس Project را انتخاب کنید . در صفحه باز شده از قسمت Database گزینه Visual C# SQL CLR Database Project را انتخاب کرده و پس از انتساب نامی به آن ، بر روی Ok کلیک نمائید.



حال صفحه ای باز می شود و باید در قسمت Server name ، نام سرور را وارد نمائید پس از آن در قسمت Connect to a database ، پایگاه داده مورد نظر را انتخاب کرده و بر روی Ok کلیک کنید.

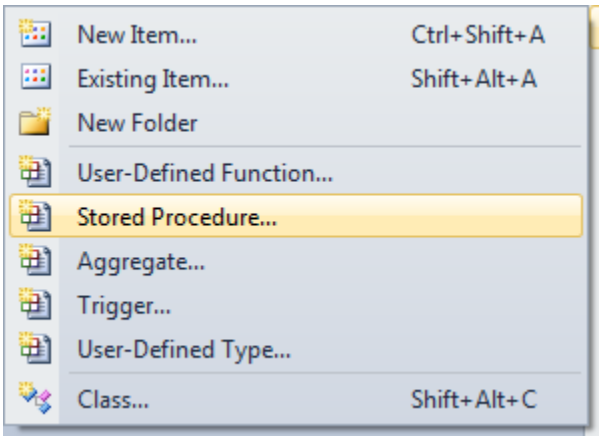


پس از آن صفحه ای باز شده و از شما سوال می کند که می خواهید clr debugging فعال شود یا خیر ؟

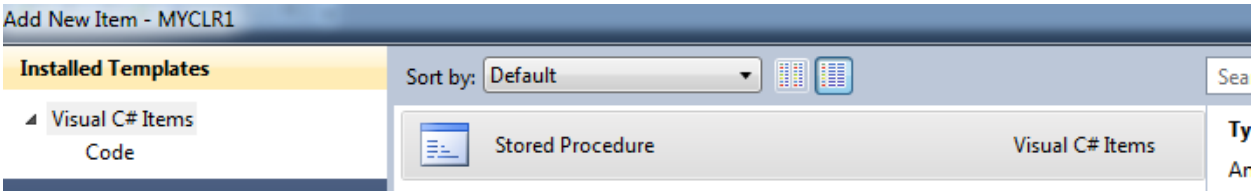


در این جا ما بر روی No کلیک می کنیم.

بر روی نام clr کلیک راست کرده و گزینه Add و سپس ...Stored Procedure را انتخاب کنید.



در صفحه باز شده گزینه Stored Procedure را انتخاب کرده و پس از انتساب نامی به آن بر روی Ok کلیک کنید.



حال می توان Procedure ای برای بانک اطلاعاتی بنویسید .

برای مثال ما در Visual Studio قطعه کدهای زیر را نوشتیم.

```
public partial class StoredProcedures
{
    [Microsoft.SqlServer.Server.SqlProcedure]
    public static void SELDATA()
    {
        SqlConnection CONN = new SqlConnection("CONTENXT CONNNECTION =TRUE");
        SqlCommand COM = CONN.CreateCommand();
        COM.CommandText = "SEELCT * FROM EMPLOYEE";
        CONN.Open();
        COM.ExecuteNonQuery();
        CONN.Close();
    }
};
```

پس از آن باید این قطعه کد ها را کامپایل کنیم . (ابتدا F6 را زده و سپس SHIFT+F6 را فشار دهید.)

حال از مسیر برنامه در پوشه : \bin\Debug ... ، مسیر فایل دارای فایل dll را کپی کنید.

سپس در SQL SERVER از طریق رویه سیستمی SP_CONFIGURE باید CLR را در SQL فعال کنیم.

```
EXEC sp_configure 'CLR_ENABLED',1
RECONFIGURE
```

پس از فعال سازی CLR باید در SQL ، اسمبلی ایجاد کرد : (پروژه ای که در Visual Studio ایجاد کردید باید Framework آن 3.5 تنظیم شده باشد .)

```
CREATE ASSEMBLY CLR
FROM
'C:\Users\AMIN\Desktop\CLR\SqlServerProject1\SqlServerProject1\bin\Debug\SqlS
erverProject1.dll'
WITH PERMISSION_SET=SAFE
```

عبارت Permission_set=safe همیشه باید مقداردهی شود . از این عبارت برای تعیین مجوزهای دسترسی به اسمبلی استفاده می شود.

پس از مراحل فوق ، حال می توان رویه ای از روی اسمبلی ایجاد نمود

```
CREATE PROCEDURE SP_DATA
AS EXTERNAL NAME CLR.StoredProcedures.SELDATA
```

- **آشنایی با برخی از رویه های سیستمی**
  - SP_TABLES : اطلاعاتی درباره همه جداول و نماهای بانک اطلاعاتی می دهد.
  - SP_PKEYS : اطلاعات کلید اصلی یک جدول را نمایش می دهد.
  - SP_STORED_PROCEDURES : اطلاعات همه رویه های ذخیره شده را نمایش می دهد.

```
USE DB
GO
EXEC sp_tables
EXEC sp_stored_procedures
EXEC sp_pkeys DEPARTMENT
```

- گزینه **SCAN FOR STARTUP PROCS**:

برای این که رویه هایی هنگام راه اندازی SQL SERVER به طور اتوماتیک اجرا شوند ، از این گزینه استفاده می شود :

```
EXEC sp_configure 'SHOW ADVANCED OPTION',1
RECONFIGURE
```

پس از این که این گزینه فعال شد ، حال می خواهیم رویه ای هنگام اجرای sql به طور خودکار اجرا شود ، برای این کار از رویه سیتمی SP_PROCOPTION استفاده می شود.

```
EXEC sp_procoption 'SP_DEL_EMPLOYEE','STARTUP','TRUE'
```

نکته : فقط می توان رویه های موجود در پایگاه داده MASTER را تنظیم نمود.

- ایجاد **UDF** و اجرای آن

توضیح این بخش را با یک مثال نشان می دهیم . برای ایجاد یک UDF از دستور Create Function استفاده می شود . پس از آن باید نامی به تابع نسبت داد.

تابع زیر فاقد پارامتر ورودی است . اگر می خواهید تابع دارای پارامتر ورودی باشد ، کافی است آن ها را در داخل پرانتز وارد نمائید.

پس از پرانتز باز و بسته باید نوع خروجی را مشخص نمود . هر نوع استاندارد از جمله نوع table نیز می توان در نظر گرفت .

از عبارت With Encryption نیز می توان برای رمزنگاری تابع نیز استفاده نمود.

دستورات تابع در داخل بلوک begin/end قرار می گیرد و در داخل آن فقط می توان :

- دستورات انتساب
- دستورات declare
- دستورات کنترل جریان
- دستورات Select , Insert,Update , Delete استفاده نمود.

```
CREATE FUNCTION GET_COUNT()
RETURNS INT
WITH ENCRYPTION
BEGIN
DECLARE @C INT
SELECT @C=  COUNT(*) FROM EMPLOYEE
RETURN @C
END
```

دستور آخر در بلوک begin/end باید دستور Return باشد .

- مثال : تعریف تابعی که نام و نام خانوادگی را برمی گرداند با پارامتر ورودی شماره کارمندی .

```
CREATE FUNCTION GETNAME(@EMP_NO INT)
RETURNS NVARCHAR(50)
BEGIN
DECLARE @NAME NVARCHAR(50)
SELECT @NAME= EMP_FNAME + EMP_LNAME FROM EMPLOYEE WHERE EMP_NO=@EMP_NO
RETURN @NAME
END
```

- مثال : تعریف تابعی که مشخصات کارمندان را براساس پارامتر ورودی آن که شماره اداره است را نشان دهد.

```
CREATE FUNCTION GETEMPLOYEE(@DEPT_NO INT)
RETURNS TABLE AS RETURN
(SELECT * FROM EMPLOYEE WHERE DEPT_NO=@DEPT_NO)
```

برای اجرای تابع :

```
SELECT * FROM DBO.GETEMPLOYEE(4)
```

نکته ای که باید یادآور شد ، می توان تابع را با جداول دیگر پیوند داد . برای مثال :

```
SELECT * FROM dbo.GETEMPLOYEE(4)AS T1 INNER JOIN DEPARTMENT
ON T1.DEPT_NO = DEPARTMENT.DEPT_NO
```

	EMP_NO	EMP_FNAME	EMP_LNAME	DEPT_NO	DEPT_NO	DEPT_NAME	LOCATION
1	104	MARYAM	JAHEDI	4	4	COMPUTER	SHIRAZ
2	105	REZA	RASOULI	4	4	COMPUTER	SHIRAZ

فصل پنجم – شاخص ها

در این فصل در مورد شاخص ها و نقش آن ها در بهینه سازی پاسخ پرس و جو ها بحث می شود.

به طور کلی پایگاه داده از شاخص ها برای **فراهم سازی دسترسی سریع** به داده های رابطه ای استفاده می کند. شاخص به صورت یک ساختار فیزیکی مجزا است که توانایی پرس و جو ها را برای دسترسی سریع به یک یا چند سطر داده بالا می برد.

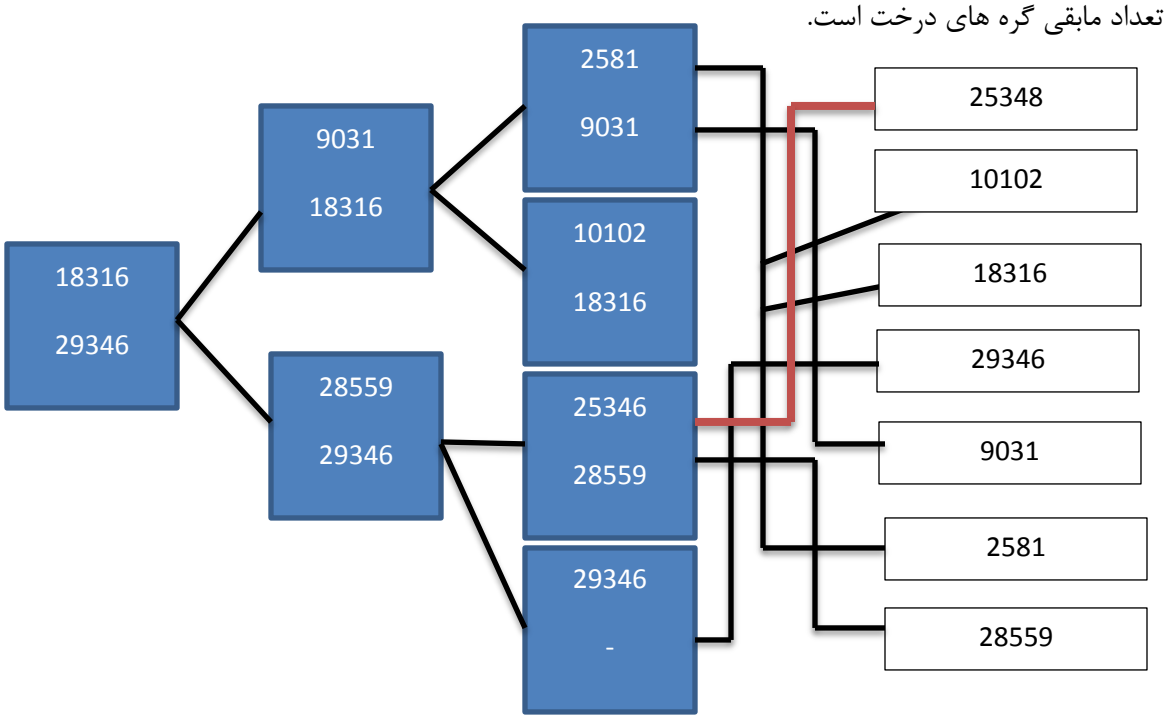
شاخص ها از بسیاری از لحاظ شبیه فهرست کتاب است . زمانی که به دنبال یک موضوع در داخل کتاب هستید ، فهرست کتاب را برای یافتن صفحات موضوع جستجو می کنید و در پایگاه داده ، وقتی به دنبال یک سطری از جدولی هستید ، پایگاه داده از یک شاخص برای یافتن محل فیزیکی سطر استفاده می کند.

فهرست کتاب و شاخص پایگاه داده دارای دو تفاوت اصلی هستند :

- شما به عنوان خواننده می توانید تصمیم بگیرید که از فهرست استفاده کنید یا نه اما در بانک اطلاعاتی ممکن نیست و بهینه ساز پرس و جو تصمیم می گیرد که از شاخص استفاده کند یا نه .
- فهرست کتاب همیشه ثابت است ولی شاخص داده ها همواره در حال تغییر است .

شاخص ها در ساختمان داده های اضافی به نام صفحات شاخص ذخیره می شوند. به ازای هر سطر شاخص بندی شده ، یک ورودی شاخص در صفحه شاخص ذخیره می شود. هر ورودی شاخص هم شامل یک کلید شاخص و یک اشاره گر می باشد.

شاخص های Database Engine با استفاده از ساختار B Tree + ایجاد می شوند . در این ساختار تعداد برگ های آن مساوی تعداد مابقی گره های درخت است.



همان طور که در شکل بالا ملاحظه می کنید ، برای مثال به دنبال مقدار ۲۵۳۴۸ هستیم . ابتدا از ریشه جستجو آغاز می شود . سپس مقادیر مساوی و بزرگتر از مقدار ۲۵۳۴۸ را می یابیم. پس مقدار ۲۹۳۴۶ بازیابی می شود ، سپس مقدار ۲۸۵۵۹ و بعد از آن ۲۵۳۴۸ بازیابی می شود.

شاخص های خوشه ای

ترتیب فیزیکی داده های جدول را تعیین می کند . برای هر جدول یک شاخص خوشه ای می توان در نظر گرفت . زمانی که کلید اصلی برای یک جدول تعیین می کنید ، به طور پیش فرض برای آن جدول یک شاخص خوشه ای ایجاد می شود. اگر شاخص غیر خوشه ای را بر روی یک ستون غیر منحصر به فرد ایجاد کنید ، پایگاه داده برای این که منحصر به فردی را حفظ نماید ، یک شماره ۴ بایتی به سطر هایی که مقدار تکراری دارند ، می افزاید.

شاخص غیر خوشه ای

ساختار آن همانند ساختار شاخص خوشه ای است اما با این تفاوت که :

- شاخص غیرخوشه ای ترتیب فیزیکی داده های جدول را تغییر نمی دهند.
- صفحات برگ هم شامل یک کلید و نشانه هستند.

اگر جدولی شامل یک شاخص خوشه ای باشد ، نشانه در شاخص غیرخوشه ای به ساختار B Tree اشاره می کند.

اگر جدول شامل یک شاخص غیر خوشه ای باشد ؛ نشانه شامل : ۱- آدرس فایلی است که جدول در آن قرار دارد ۲- آدرس صفحه ۳- آدرس سطر موجود در صفحه شاخص

ایجاد شاخص

```
CREATE UNIQUE CLUSTERED|NONCLUSTERED INDEX INDEX_NAME
ON TABLE_NAME(COLUMN_NAME)
INCLUDE(COLUMN_NAME)
WITH
(
FILLFACTOR=N,
PAD_INDEX=ON|OFF,
ALLOW_ROW_LOCKS=ON|OFF,
ALLOW_PAGE_LOCKS=ON|OFF,
```

```
DROP_EXISTING=ON|OFF,
SORT_IN_TEMPDB=ON|OFF,
STATISTICS_NORECOMPUTE=ON|OFF
)
```

برای ایجاد یک شاخص منحصربه فرد از عبارت UNIQUE در ساخت شاخص استفاده می شود.

برای مشخص کردن شاخص به صورت خوشه ای از CLUSTERED و برای مشخص کردن به صورت غیرخوشه ای از NONCLUSTERED استفاده می شود.

در صورتی که مشخص نکنید ، شاخص شما خوشه ای یا خیرخوشه ای باشد ، پایگاه داده آن را به صورت غیرخوشه ای در نظر می گیرد.

حداکثر روی یک جدول می توان ۲۹۴ شاخص غیرخوشه ای ایجاد کرد.

شاخص می تواند منفرد باشد یعنی در بخش CLOUMN_NAME می توان فقط نام یک ستون را به کار برد و هم چنین شاخص می تواند ترکیبی باشد ، یعنی در بخش ذکر شده می توان نام چند ستون را افزود.

برای تعیین میزان خالی بودن صفحات برگ شاخص از FILLFACTOR استفاده می شود . اگر مقدار ۱۰۰ به آن دهید یعنی هر صفحه شاخص پر بوده و دیگر مکانی برای درج داده های جدید نیست. اگر صفر به این پارامتر دهید ، یعنی در صفحات میانی شاخص ، یک مکان خالی برای درج سطر جدید وجود دارد.

برای تعیین میزان خالی بودن صفحات میانی از PAD_INDEX استفاده می شود.

برای این که میزان بهره وری شاخص غیرخوشه ای موجود روی یک جدول بالا برود ، در هنگامی که یک شاخص غیرخوشه ای روی جدول موجود است و بخواهیم شاخص غیرخوشه ای دیگری به آن جدول اضافه کنیم از DROP_EXISTING استفاده می شود ، هم چنین از این پارامتر برای ایجاد مجدد شاخص نیز استفاده می شود.

اگر بخواهیم در هنگام ایجاد شاخص ، داده ها در پایگاه داده سیستمی TEMP قرار گیرند ، از SORT_IN_TEMPDB استفاده می شود.

اگر بخواهیم درج مقادیر تکراری در شاخص را نادیده بگیریم از IGNORE_DUP_KEY استفاده می کنیم.

برای قفل هر سطر از شاخص از ALLOW_ROW_LOCKS و برای قفل هر صفحه شاخص از ALLOW_PAGE_LOCKS استفاده می شود.

برای ذخیره آمار مربوط به شاخص از STATISTICS_NORECOMPUTE نیز استفاده می شود.

مثال : ایجاد یک شاخص غیرخوشه ای بر روی جدول EMPLOYEE و روی ستون EMP_NO :

```
CREATE NONCLUSTERED INDEX IDX1
ON EMPLOYEE(EMP_NO)
```

مثال : ایجاد شاخص روی جدول WORK_ON به صورت ترکیبی روی ستون های EMP_NO و PRJ_NO طوری که میزان خالی بودن صفحات برگ ۸۰ درصد باشد :

```
CREATE NONCLUSTERED INDEX IDX_WORKS_ON
ON WORK_ON (EMP_NO,PRJ_NO)
WITH(FILLFACTOR=80)
```

به دست آوردن اطلاعات از شاخص های ذخیره شده

برای بدست آوردن اطلاعات شاخص ها از رویه سیستمی SP_HELPINDEX استفاده می شود که اطلاعات شاخص های موجود روی هر جدول را نشان می دهد. برای بدست آوردن اطلاعات قطعه بندی هر شاخص هم از SYS.INDEXS استفاده می شود :

```
SELECT * FROM SYS.indexes WHERE index_id IN (SELECT index_id FROM SYS.indexes
WHERE SYS.indexes.name='IDX1')
```

اطلاعات راجع به شاخص IDX1 را نشان می دهد.

حال می خواهیم متوجه شویم روی جدول WORK_ON چه شاخص هایی موجود است :

```
EXEC sp_helpindex 'DBO.WORK_ON'
```

نکته ای که باید ذکر نمود این است دو فرم قطعه بندی برای شاخص ها وجود دارد :

- داخلی : میزان داده های ذخیره شده برای هر شاخص مشخص باشد.
- خارجی : ترتیب منطقی صفحات اشتباه باشد.

تغییر شاخص

برای تغییر شاخص از ALTER INDEX استفاده می شود. می توان در هنگام تغییر شاخص آن ها را

۱- غیرفعال نمود. (با استفاده از DISABLE)
۲- دوباره سازماندهی نمود. (با استفاده از REORGANIZE)
۳- دوباره ایجاد کرد. (با استفاده از REBULID)

تغییر شاخص و غیر فعال نمودن آن :

```
ALTER INDEX IDX1
ON WORKS_ON
DISABLE
```

تغییر شاخص و دوباره ایجاد نمودن آن :

```
ALTER INDEX IDX1
ON WORKS_ON
REBUILD WITH(FILLFACTOR=60)
```

حذف شاخص

```
DROP INDEX IDX1 ON WORKON
```

شاخص پوششی

زمانی یک شاخص پوششی است که در پرس و جو تمام ستون های جدول به کار رود .

```
CREATE NONCLUSTERED INDEX IDX3
ON EMPLOYEE(EMP_NO)
INCLUDE(EMP_FNAME,EMP_LNAME,DEPT_NO)
```

شاخص FULLTEXT:

برای پرس و جوی پیچیده روی داده های کارکتری از شاخص FULLTEXT استفاده می شود.

مراحل ایجاد این شاخص به این صورت است :

۱- ایجاد یک کاتولوگ

۲- ایجاد شاخص FULLTEXT

مثال : جدولی با نام POST می سازیم و می خواهیم روی داده های کارکتری آن یک شاخص ایجاد کنیم :

```
CREATE TABLE POST
(
POST_ID INT,
POST_NAME NVARCHAR(30),
[TEXT] NTEXT
)
CREATE UNIQUE CLUSTERED INDEX IDX1 ON POST(POST_ID)
CREATE FULLTEXT CATALOG FT_POST
CREATE FULLTEXT INDEX ON POST([TEXT]) KEY INDEX IDX1
```

ابتدا یک جدول ساختیم و روی این جدول یک شاخص منحصر به فرد ، خوشه ای ساختیم . بعد از آن حال می توانیم شاخص FULLTEXT روی جدول ایجاد نمائیم.

اجرای شاخص FULLTEXT:

```
INSERT INTO POST VALUES
(1,N'جمله اولین',N'این جمله اولین و است آزمایشی جمله است')
(2,N'جمله دومین',N'این جمله دوم آزمایشی است')
(3,N'جمله سومین',N'این جدول سومین آزمایشی است.')
```

ابتدا برای این که بهتر مشخص شود ، تعدادی سطر به جدول می افزاییم . حال نوبت به جستجو می رسد :

```
SELECT * FROM POST
WHERE FREETEXT([TEXT],N'دومین')
```

همچنین می توان در پرس و جو از عبارات منطقی نیز استفاده نمود :

```
SELECT * FROM POST
WHERE CONTAINS([TEXT],N'"دومین" AND "سومین"')
```

تغییر شاخص FULLTEXT:

برای حذف یک ستون یا افزودن یک ستون از شاخص FULLTEXT ، باید آن را ویرایش کرد :

```
ALTER FULLTEXT INDEX IDX1 ON POST
ADD(POST_NAME)
DROP([TEXT])
```

تفاوت های بین شاخص FULLTEXT و شاخص های عادی

شاخص FULLTEXT	شاخص عادی
این شاخص ها گروه بندی می شوند.	گروه بندی در این شاخص ها وجود ندارد.
فقط می توان یک شاخص FULLTEXT روی یک جدول ایجاد کرد.	بیش از یک شاخص عادی می توان روی یک جدول ایجاد نمود.
در سیستم فایل ذخیره شده و توسط پایگاه داده مدیریت می شود.	در پایگاه داده ذخیره می شود.

فصل ششم – نماها

نما چیست ؟

نما اشیایی از پایگاه داده هستند که با استفاده از اطلاعات فراداده (از یک یا چند جدول) مشتق می شوند.

داده های نما روی دیسک ذخیره نمی شود. بلکه نام نما و روش های بازیابی سطرها از جدول ، بر روی دیسک ذخیره می شوند.

نحوه ایجاد یک نما

نحوه ایجاد را با یک مثال نشان می دهیم . می خواهیم روی جدول EMPLOYEE یک نما ایجاد کنیم:

```
CREATE VIEW VIEW_EMPLOYEE(EMP_NO,EMP_FNAME,EMP_LNAME,DEPT_NO)
WITH ENCRYPTION,SCHEMABINDING,VIEW_METADATA
AS
SELECT EMP_NO,EMP_FNAME,EMP_LNAME,DEPT_NO FROM EMPLOYEE
WITH OPTION CHECK
GO
```

فراخوانی نما همانند جدول است :

```
SELECT * FROM VIEW_EMPLOYEE
```

با افزودن ENCRYPTION می توان دستور SELECT را به صورت رمز شده ذخیره کرد تا امنیت پایگاه داده بالا رود.

با SCHEMABINDING هم ، نما را مقید به شِما می سازید. یعنی در هنگام SELECT باید نام اشیا را به صورت دو بخشی نوشت .

با VIEW _METADATA هم می توان در آینده با استفاده از تریگر ها ، ستون های به کار رفته در نما را به روز کرد.

با OPTION CHECK هم می توان هنگامی که داریم سطری به نما می افزاییم ، بررسی می کند که شرط WHERE به کار رفته درست باشد .

اهداف استفاده از نماها

- محدود کردن کاربرد های ستون / سطر های یک جدول

- برای پنهان کردن جزئیات پرس و جو
- برای محدود کردن مقادیر درج شده یا به روز شده

مثال : نمایی بر روی جدول WORK_ON ایجاد کرده که رکوردهایی که شغل آن ها آنالیزور است را نمایش دهد.

```
CREATE VIEW VIEW_WORK
AS
SELECT * FROM WORKON WHERE JOB=N'آنالیزور'
GO
```

به کار بردن نام ستون ها اختیاری است اما در بعضی از مواقع باید نام ستون را نوشت : زمانی که از عبارات یا توابع تجمعی استفاده می شود .

```
CREATE VIEW V_WORK(PRJ_NO,COUNT_PRJ)
AS
SELECT PRJ_NO,COUNT(*) FROM WORKON
GROUP BY PRJ_NO
```

همین مثال را در صورتی که برای COUNT(*) ، نامی مستعار انتخاب شود ، دیگر نیازی به نوشتن نام ستون ها نیست :

```
CREATE VIEW V_WORK
AS
SELECT PRJ_NO,COUNT(*)AS COUNT_PRJ FROM WORKON
GROUP BY PRJ_NO
```

نکته : می توان یک نما را از نمایی دیگر نیز مشتق کرد :

```
CREATE VIEW V_WORK3
AS
SELECT EMP_NO FROM VIEW_WORK WHERE EMP_NO=101
GO
```

اگر جدولی را حذف کنید ، نماهای مربوط به آن حذف نخواهند شد.

تغییر نما

می خواهیم نمای مثال قبل را ویرایش کنیم ، طوری که علاوه بر EMP_NO ، PRJ_NO هم در پرس و جو شرکت داشته باشد :

```
ALTER VIEW V_WORK3
AS
SELECT EMP_NO,PRJ_NO FROM VIEW_WORK WHERE EMP_NO=101
GO
```

حذف نما

```
DROP VIEW VIEW_NAME
```

نکته : قبلا نمایی را ایجاد نموده اید و حال می خواهید آن را ویرایش کنید ، اما کد های به کار رفته در آن را از یاد برده اید برای این کار باید از رویه سیستمی SP_HELPTEXT کمک بگیرید

```
EXEC sp_helptext V_WORK3
```

Results | Messages

	Text
1	CREATE VIEW V_WORK3
2	AS
3	SELECT EMP_NO,PRJ_NO FROM VIEW_WORK WHERE EMP_NO...

درج در نما

زمانی که سطری را در یک نما درج می کنید ، در واقع آن در جدول اصلی درج خواهد شد.

```
CREATE VIEW V_DEPART
AS
SELECT DEPT_NO,DEPT_NAME FROM DEPARTMENT
GO
```

می خواهیم در نمای بالا ، رکورد جدید را درج کنیم :

```
INSERT INTO V_DEPART VALUES('ACCOUNTING')
```

علت این که DEPT_NO را ذکر نکردیم ، این است که آن به صورت IDENTITY می باشد.

مثال : نمای بالا را ویرایش می کنیم طوری که فقط سطرهایی را نمایش دهد که نام آن ها ACCONTING باشد . در ضمن CHECK OPTION را نیز تنظیم می کنیم.

```
ALTER VIEW V_DEPART
AS
SELECT DEPT_NO,DEPT_NAME FROM DEPARTMENT
WHERE DEPT_NAME='ACCOUNTING'
WITH CHECK OPTION
GO
```

حال می خواهیم در این نما عمل درج را انجام دهیم :

```
INSERT INTO V_DEPART VALUES('COMPUTER')
```

با خطا روبرو خواهید شد ، چون CHECK OPTION ابتدا شرط موجود در نما را بررسی می کند و چون شرط با چیزی که می خواهیم درج کنیم ، یکی نیست ، عمل درج صورت نمی گیرد.

اعمال به روز رسانی و حذف هم بر روی نماها قابل اجرا هستند.

نماهای شاخص دار

اگر می خواهید بر روی یک نما ، شاخص ایجاد نمائید ، باید SCHEMABINDING را روی نما تنظیم کرده باشید.

فصل هفتم – سیستم امنیت DATABASE ENGINE

سیستم امنیت پایگاه داده شامل دو زیر سیستم است :

- امنیت ویندوز (Windows)
- امنیت Sql Server

امنیت ویندوز در سطح سیستم عامل است. تصدیق از طریق حساب های کاربری که کاربران با آن ها با ویندوز لاگین کرده اند .

امنیت Sql Server در سطح سیستم است. یعنی کاربران بعد از ورود به سیستم عامل ، Sql Server یک حساب کاربری برای آن ها تعریف می کند که کاربر باید با آن ها ورود نماید.

Database Engine از طریق این دو زیر سیستم امنیتی ، عمل تصدیق را انجام می دهد :

- Windows Mode : پایگاه داده فرض می کند ، حساب کاربری مورد نظر توسط سیستم عامل پذیرفته شده است. (از حساب های کاربری برای ورود به سیستم پایگاه داده استفاده می شود.)
- Mixed Mode : در این روش ، بخشی از حساب های کاربری توسط زیر سیستم امنیتی ویندوز و بخشی توسط Sql Server تنظیم می شوند.

تنظیم با مد Mixed

```
EXEC XP_INSTANCE_REGWRITE
N'HKEY_LOCAL_MACHINE',N'SOFTWARE\Microsoft\MSSQLServer\MSSQLServer',
N'LOGINMODE',REG_DWORD,2
```

تنظیم با مد Windows

```
EXEC XP_INSTANCE_REGWRITE
N'HKEY_LOCAL_MACHINE',N'SOFTWARE\Microsoft\MSSQLServer\MSSQLServer',
N'LOGINMODE',REG_DWORD,1
```

رمزنگاری داده ها

اگر بخواهید برای رمزنگاری و رمزگشایی داده ها از کلید و گواهی استفاده نکنید ، می توانید از توابع

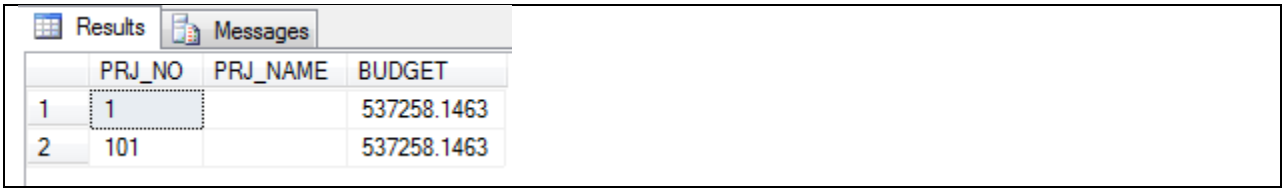
- EncryptByPassPhrase
- DecryptByPassPhrase

این دو تابع از طریق یک کلمه عبور داده های مورد نظر را رمزنگاری و رمزگشایی می کنند.

مثال : می خواهیم نام پروژه ها در جدول Project را رمزنگاری و سپس از حالت رمز خارج کنیم.

```
UPDATE PROJECT
SET PRJ_NAME=ENCRYPTBYPASSPHRASE('123',PRJ_NAME)
```

اگر سطرهای جدول را مشاهده نمائید ، در ستون Prj_name محتوایی را مشاهده نخواهید کرد.



Results | Messages

	PRJ_NO	PRJ_NAME	BUDGET
1	1		537258.1463
2	101		537258.1463

حال می خواهیم جدول را از حالت رمزنگاری خارج کنیم :

```
UPDATE PROJECT
SET PRJ_NAME=DECRYPTBYPASSPHRASE('123',PRJ_NAME)
```

شاه کلید سرویس ،درواقع کلیدی است که بر تمام کلیدها و گواهی های دیگر تسلط دارد. زمانی که DataBase Engine را نصب می کنید ، شاه کلید سرویس به طور خودکار ایجاد می شود.

پشتیبان گیری از شاه کلید سرویس

```
BACKUP SERVICE MASTER KEY
TO FILE='D:\DATABASE\TEST.BAK'
ENCRYPTION BY PASSWORD='123'
```

بازگردانی پشتیبان شاه کلید سرویس

```
RESTORE SERVICE MASTER KEY
FROM FILE='D:\DATABASE\TEST.BAK'
```

```
DECRYPTION BY PASSWORD='123'
```

شاه کلید پایگاه داده ، یک شی رمزنگاری ریشه برای رمزنگاری کلید ها و گواهی های سطح پایگاه داده است. توسط دستور Create Master Key ایجاد می شود. چون شاه کلید پایگاه داده ، زیر مجموعه شاه کلید سرویس است ، پس می تواند آن را رمزگشایی نماید.

ایجاد ، تغییر و حذف شاه کلید پایگاه داده

زمانی که یک شاه کلید پایگاه داده ایجاد می کنید. یک لایه اضافی امنیت برای رمزنگاری کلیدهای نامتقارن و گواهی ها اضافه می شود.

ایجاد شاه کلید پایگاه داده

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '123'
```

شاه کلید پایگاه داده ، همانند شاه کلید سرویس بدون نام است.

ویرایش شاه کلید پایگاه داده

```
ALTER MASTER KEY
REGENERATE WITH ENCRYPTION BY PASSWORD = '1234'
```

در این هنگام ابتدا تمامی اطلاعات رمزگشایی شده و سپس با رمز جدید ، رمزنگاری می شوند.

حذف شاه کلید پایگاه داده

```
DROP MASTER KEY
```

در صورتی که آن را برای رمزنگاری به کار برده باشید ، نخواهید توانست آن را حذف نمائید.

پشتیبان گیری و بازگردانی شاه کلید پایگاه داده

```
BACKUP MASTER KEY
TO FILE = 'D:\DATABASE\MASTERKEY.BAK'
ENCRYPTION BY PASSWORD='123'
```

بازگردانی شاه کلید پایگاه داده

```
RESTORE MASTER KEY
FROM FILE = 'D:\DATABASE\MASTERKEY.BAK'
DECRYPTION BY PASSWORD='123'
ENCRYPTION BY PASSWORD='1234'
```

زمانی که یک شاه کلید پایگاه داده ایجاد می کنید، شاه کلید پایگاه داده توسط شاه کلید سرویس و کلمه عبور شاه کلید رمزنگاری می شود.

اگر نمی خواهید شاه کلید پایگاه داده توسط شاه کلید سرویس رمزنگاری شود می توانید آن را از پایگاه داده حذف و یا اضافه نمائید:

حذف شاه کلید سرویس از رمزنگاری شاه کلید پایگاه داده

```
ALTER MASTER KEY
DROP ENCRYPTION BY SERVICE MASTER KEY
```

اضافه کردن شاه کلید سرویس به رمزنگاری شاه کلید پایگاه داده

```
OPEN MASTER KEY DECRYPTION BY PASSWORD='123'
ALTER MASTER KEY
ADD ENCRYPTION BY SERVICE MASTER KEY
CLOSE MASTER KEY
```

چون شاه کلید سرویس حذف شده بود ، برای دسترسی به شاه کلید پایگاه داده ، باید شاه کلید پایگاه را ابتدا باز و سپس ببندید.

کلیدهای متقارن

سیستم رمزنگاری وقتی از کلید متقارن استفاده می کند ، کلید رمزنگاری و رمزگشایی مشترک است. کلیدهای متقارن دارای دو مزیت هستند : اول این که حجم زیادی از داده ها را می توانند رمزنگاری کنند و دوم این که سرعت بیشتری نسبت به کلیدهای نامتقارن دارند.

عیب این کلیدها در این است که در سیستم های توزیع شده ، نا امن هستند.

ایجاد یک کلید متقارن

```
CREATE SYMMETRIC KEY KEY10
WITH ALGORITHM = DES
```

```
ENCRYPTION  BY PASSWORD='123'
```

KEY10 بیانگر نام کلید متقارن است. در قسمت WITH ALGORITHM می توان الگوریتم رمزنگاری کلید را مشخص نمود .(DES- TRIPLE_DES - DESX و ...) . در آخر کلید را با پسوردی محافظت می کنیم.(مکانیزم محافظت از کلید)

حذف یک کلید متقارن

```
DROP SYMMETRIC KEY KEY10
```

نکته : قبل از استفاده از کلید متقارن برای رمزنگاری داده ها ، باید کلید را باز کنید : Open Symmetric Key

بعد از باز کردن لازم است با استفاده از EncrypteByKey ، داده ها را رمزنگاری نمائید. با استفاده از DecrypeByKey از حالت رمزنگاری خارج کنید.

نمایش کلیدهای متقارن پایگاه داده

```
SELECT * FROM SYS.symmetric_keys
```

تغییر روش رمزنگاری یک کلید متقارن

```
OPEN SYMMETRIC KEY KEY10
DECRYPTION BY PASSWORD='123'

ALTER SYMMETRIC KEY KEY10
ADD ENCRYPTION BY PASSWORD='1234'

CLOSE SYMMETRIC KEY KEY10
```

رمزنگاری داده ها با استفاده از کلید متقارن

```
OPEN SYMMETRIC KEY KEY10
DECRYPTION BY PASSWORD='123'

UPDATE PROJECT
SET PRJ_NAME=ENCRYPTBYKEY(KEY_GUID('KEY10'),PRJ_NAME);
```

```
CLOSE SYMMETRIC KEY KEY10
```

Key_Guid یک شناسه منحصر به فرد جهانی کلید متقارن است.

رمزگشایی داده ها با استفاده از DecryptByKey :

```
OPEN SYMMETRIC KEY KEY10

DECRYPTION BY PASSWORD='123'
UPDATE PROJECT
SET PRJ_NAME=DECRYPTBYKEY(PRJ_NAME);

CLOSE SYMMETRIC KEY KEY10
```

کلید های نامتقارن

اگر در یک محیط توزیع شده هستید و یا کلید متقارن از امنیت لازم برخوردار نیست می توانید از کلید نامتقارن استفاده کنید. کلید نامتقارن شامل یک کلید خصوصی و یک کلید عمومی متناظر است.

برای رمزنگاری داده ها با استفاده از کلید نامتقارن می توانید از تابع EncryptByAsymkey استفاده کنید.

ایجاد یک کلید نامتقارن

```
CREATE ASYMMETRIC KEY KEY9
WITH ALGORITHM = RSA_512
ENCRYPTION BY PASSWORD='123'
```

مشاهده کلیدهای نامتقارن

```
SELECT * FROM SYS.asymmetric_keys
```

تغییر کلید خصوصی کلید نامتقارن

```
ALTER ASYMMETRIC KEY KEY9
WITH PRIVATE KEY
(ENCRYPTION BY PASSWORD='1234',DECRYPTION BY PASSWORD='123')
```

رمزنگاری با استفاده از کلید نامتقارن

برای رمزنگاری با کلید نامتقارن باید از تابع EncryptByAsymKey استفاده می شود.

```
INSERT INTO PROJECT VALUES
(ENCRYPTBYASYMKEY(ASYMKEY_ID('KEY9'),'PROJECT1'),1000000)
```

رمزگشایی کلید نامتقارن

```
SELECT PRJ_NO,
CAST(DECRYPTBYASYMKEY(ASYMKEY_ID('KEY9'),PRJ_NAME,N'1234' ) AS NVARCHAR(50))
FROM PROJECT
```

گواهی ها

یک درخواست دیجیتالی امضا شده است که مقدار یک کلید عمومی را به شناسه یک فرد ، سازمان یا سرویسی که کلید خصوصی متناظر آن را نگه داشته ، مقید می سازد. مزیت اصلی آن در این است که از نگهداری تعدادی رمز برای تک تک موضوع ها آسوده است.

```
CREATE CERTIFICATE CERT1
ENCRYPTION BY PASSWORD='123'
WITH SUBJECT='VALID EMPLOYEE'
,START_DATE='1/1/2013',EXPIRY_DATE='1/1/2014';
```

نمایش گواهی های پایگاه داده

```
SELECT * FROM SYS.certificates
```

پشتیبان گیری و بازگردانی یک گواهی

```
BACKUP CERTIFICATE CERT1 TO FILE='D:\DATABASE\CERT1.BAK'
WITH PRIVATE KEY
(
FILE='D:\DATABASE\CERT1PK.BAK',
ENCRYPTION BY PASSWORD='1234',
DECRYPTION BY PASSWORD='123'
);
```

برای بازگردانی گواهی باید CERTIFICATE قبلی را حذف کنید

```
DROP CERTIFICATE CERT1

CREATE CERTIFICATE CERT1 FROM FILE='D:\DATABASE\CERT1.BAK'
WITH PRIVATE KEY
(
FILE='D:\DATABASE\CERT1PK.BAK',
DECRYPTION BY PASSWORD='1234',
ENCRYPTION BY PASSWORD='12345');
```

مدیریت کلید خصوصی گواهی

می توان کلید خصوصی یک گواهی را حذف یا اضافه کرد.

```
ALTER CERTIFICATE CERT1
REMOVE PRIVATE KEY
```

افزودن کلید خصوصی از روی فایل پشتیبان

```
ALTER CERTIFICATE CERT1
WITH PRIVATE KEY
(
FILE='D:\DATABASE\CERT1PK.BAK',
DECRYPTION BY PASSWORD='1234',
ENCRYPTION BY PASSWORD='1234'
);
```

رمزنگاری و رمزگشایی توسط گواهی ها

برای رمز نگاری از تابع EncryptByCert(certificate_id('cerrtificate_name'),data)

برای رمزگشایی هم از تابع

DecryptByCert(certificate_id('certificate_name'),data,cert_password

استفاده می شود.

```
UPDATE EMPLOYEE
SET EMP_FNAME=ENCRYPTBYCERT(CERT_ID('CERT1'),EMP_FNAME);
```

```
UPDATE EMPLOYEE
SET EMP_FNAME=DECRYPTBYCERT(CERT_ID('CERT1'),EMP_FNAME,'1234');
```

فصل هشتم – تریگر ها

زمانی که یک عمل خاص روی یک جدول یا هر شی از پایگاه داده رخ می دهد ، تریگر ها احضار می شوند. هر تریگر سه بخش دارد : **نام ، عمل و اجرا**

نام تریگر حداکثر می تواند ۱۲۸ کارکتر داشته باشد و می تواند روی دستورات DDL و یا DML عمل کند.

ایجاد تریگر DML

```
CREATE TRIGGER TRIGGER_NAME
ON TABLE_NAME FOR | AFTER | INSTEAD OF { INSERT | UPDATE | DELETE }
AS
...
```

برای ساخت تریگر از دستور Create Trigger استفاده می شود. سپس نامی را برای آن در نظر می گیریم. که همانطور که بیان شد می تواند ۱۲۸ کارکتری باشد. پس از انتساب نام به تریگر ، نوبت به آن رسیده ، تا مشخص کنیم تریگر مربوط به چه جدولی یا شی از پایگاه داده است که آن را با On مشخص می کنیم. اگر بخواهیم تریگر هنگام عمل روی شی ، احضار شود از گزینه FOR و اگر بخواهیم بعد از عمل احضار شود از گزینه AFTER استفاده می شود. وقتی بخواهیم تریگر به جای یک عمل متناظر انجام شود از INSTEAD OF استفاده می شود. گزینه های FOR و AFTER روی جداول کار می کنند درحالی که گزینه INSTEAD OF هم روی جداول و هم روی نماها قابل اجراست.

اگر در دستور ساخت تریگر از INSTER استفاده کنید ، سطرهای درج شده در Inserted قرار می گیرند. اگر بخواهید در دستور ساخت تریگر از Delete استفاده کنید ، سطرها در deleted درج خواهند شد . هنگام استفاده از عمل Update در ساخت تریگر ، ترکیب Inserted و Deleted است.

مثال :می خواهیم هنگامی که عمل درج انجام می شود ، بررسی کند اگر نامی که می خواهیم درج شود ، وجود داشت ، پیغام 'این نام وجود دارد' را چاپ نماید.

```
CREATE TRIGGER CHECK_ON_INSERT_DEPARTMENT
ON DEPARTMENT FOR  INSERT
AS
IF  ( SELECT COUNT(*) FROM DEPARTMENT WHERE DEPT_NAME IN ( SELECT DEPT_NAME
FROM inserted))>1
```

```
BEGIN
PRINT 'THIS NAME IS EXISTS'
ROLLBACK TRANSACTION
END
```

اجرا :

```
INSERT INTO DEPARTMENT VALUES('COMPUTER','BIRJAND')
```

```
Messages
THIS NAME IS EXISTS
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.
```

فقط مالک پایگاه داده ، مالک جدول و مدیران ddL قادر خواهند بود ، تریگر اجرا کنند.

مثال : می خواهیم هنگامی که عمل به روز رسانی روی جدول project انجام شود ، مقادیر به روز شده در جدولی دیگر قرار گیرند.

ابتدا یک جدول با نام TBL_BUDGET می سازیم :

```
CREATE TABLE TBL_BUDGET
(
	PRJ_NO INT NULL,
	USERNAME NVARCHAR(20) NULL,
	[DATE] DATETIME NULL,
	BUDGET_OLD FLOAT NULL,
	BUDGET_NEW FLOAT NULL
)
```

سپس تریگر را روی آن اجرا می کنیم.

```
CREATE TRIGGER CHECK_UPDATE_ON_PROJECT
ON PROJECT AFTER UPDATE
AS
IF UPDATE(BUDGET)
BEGIN
DECLARE @PRJ_NO INT
DECLARE @BUD_OLD FLOAT
DECLARE @BUD_NEW FLOAT

SET @PRJ_NO = (SELECT PRJ_NO FROM deleted)
SET @BUD_OLD = (SELECT BUDGET FROM deleted)
SET @BUD_NEW = (SELECT BUDGET FROM inserted)

INSERT INTO TBL_BUDGET VALUES
(@PRJ_NO,USER_NAME(),GETDATE(),@BUD_OLD,@BUD_NEW)
END
```

حال می خواهیم تریگر بالا را تغییر داده و یک خط پیغام نیز به آن اضافه نمائیم :

```
ALTER TRIGGER CHECK_UPDATE_ON_PROJECT
ON PROJECT AFTER UPDATE
AS
IF UPDATE(BUDGET)
BEGIN
DECLARE @PRJ_NO INT
DECLARE @BUD_OLD FLOAT
DECLARE @BUD_NEW FLOAT

SET @PRJ_NO = (SELECT PRJ_NO FROM deleted)
SET @BUD_OLD = (SELECT BUDGET FROM deleted)
SET @BUD_NEW = (SELECT BUDGET FROM inserted)

INSERT INTO TBL_BUDGET VALUES
(@PRJ_NO,USER_NAME(),GETDATE(),@BUD_OLD,@BUD_NEW)
PRINT 'INSERT IS SUCCESFULLY'
END
```

حال اگر ما جدول PROJECT را به روز رسانی نمائیم :

```
UPDATE PROJECT SET
BUDGET=2000000 WHERE PRJ_NO=101
```

Messages

```
(1 row(s) affected)
INSERT IS SUCCESFULLY

(1 row(s) affected)
```

تریگر های After معمولا برای موارد زیر به کار برده می شوند :

- برای پیاده سازی عملیات حسابرسی
- پیاده سازی قواعد کاری
- ارجاع جامعیت های ارجاعی

جامعیت های ارجاعی معمولا به دو صورت مدیریت می شوند :

۱- با استفاده از دستورات Create Table و Alter Table

۲- تریگر ها

جدول EMPLOYEE و WORKON ، با هم ارتباطی از طریق EMP_NO دارند . می خواهیم تریگری بنویسیم که اگر EMP_NO در جدول WORKON تغییر کرد ، بررسی کند آیا EMP_NO جدید در EMPLOYEE موجود است یا خیر .

```
CREATE TRIGGER WORKS_ON_INTEGRITY
ON WORKON AFTER INSERT,UPDATE
AS IF UPDATE(EMP_NO)
BEGIN
IF(SELECT EMPLOYEE.EMP_NO FROM EMPLOYEE,inserted WHERE
EMPLOYEE.EMP_NO=inserted.EMP_NO)IS NULL
BEGIN
      ROLLBACK TRANSACTION
      PRINT 'NO INSERT'
END
ELSE PRINT 'ROW INSERT'
END
```

تریگر روی جدول WORKON ، بعد از عمل به روز رسانی ؛ عمل خواهد کرد. به این صورت : اگر EMP_NO به روز شد ، بررسی کن اگر در جدول EMPLOYEE و inserted ، EMP_NO برابر بودند که هیچی ، در غیر این صورت عمل را متوقف (ROLLBACK TRANSACTION) کن.

مثال : می خواهیم هنگامی که در جدول ادارات ، سطری درج شود ، عملیات درج در جدولی دیگر نیز درج شود.

```
CREATE TABLE TRACE
(
ID INT NOT NULL IDENTITY(1,1),
USERNAME VARCHAR(100),
OP VARCHAR(100) NOT NULL,
[DATE] DATETIME NOT NULL
)
GO
```

می خواهیم وقتی در جدول Departmet ، سطری درج می شود ، عمل درج با نام کاربری و تاریخ نیز در جدول بالا درج شود.

```
CREATE TRIGGER INSERT_DEPARTMENT
ON DEPARTMENT AFTER INSERT
AS
BEGIN
INSERT INTO TRACE VALUES(USER_NAME(),'INSERT',GETDATE())
PRINT 'OPERATION INSERTED INTO TRACE'
END
GO
```

نتیجه

```
INSERT INTO DEPARTMENT VALUES('OMRAN','SHIRAZ')
```

Messages

```
(1 row(s) affected)
OPERATION INSERTED INTO TRACE

(1 row(s) affected)
```

تریگر های اول و آخر

پایگاه داده این امکان را به ما می دهد که روی یک جدول یا نما ، چندین تریگر در نظر بگیریم. هم چنین می توان ترتیب اجرای تریگر ها را نیز مشخص کرد. برای این کار از رویه سیستمی sp_SetTriggerOrder استفاده می شود.

نکته : هنگامی که تریگری را ویرایش می کنید ، ترتیب تریگر ها حذف خواهد شد.

رویه ذکر شده دارای یک پارامتری است که اولویت تریگر را مشخص می کند.

- First : اولین تریگر
- Last : آخرین تریگر
- None : ترتیب مشخص شده را باطل می سازد.

```
EXEC sp_settriggerorder 'INSERT_DEPARTMENT','FIRST','INSERT'
```

می خواهیم تریکر Insert_Department ، اولین تریگر باشد.

نکته : این رویه سیستمی روی تریگر هایی عمل خواهد کرد که از گزینه After استفاده کرده باشند.

برای مشاهده این که روی جدولی ، کدام تریگر ، ابتدا عمل می کند ، کافی است از رویه سیستمی SP_HELPTERIGGER استفاده نمائید

```
EXEC sp_helptrigger 'DBO.DEPARTMENT'
```

تریگر های DDL

پایگاه داده نیز این امکان را به ما می دهد تا تریگر ها را روی دستورات DDL نیز تعریف کنیم.

هنگام ساخت تریگر DDL باید میدان تریگر نیز مشخص شود :

- Database
- All Server

دو تفاوت اصلی در تریگر های DDL و DML نیز وجود دارد : اول این که تریگر های DDL دارای میدان هستند ولی تریگر های DML فقط روی شی پایگاه داده تعریف می شوند و دوم این که تریگر های DDL از Instead OF پشتیبانی نمی کنند.

- **تریگرهای سطح پایگاه داده**

مثال : می خواهیم تریگری در سطح DataBase تعریف کنیم که اجازه ساخت جدول را به کاربران ندهد.

```
CREATE TRIGGER NOTABLE
ON DATABASE FOR CREATE_TABLE
AS
BEGIN
ROLLBACK TRANSACTION
PRINT 'NOT ALLOW CREATE TABLE'
END
```

حال اگر دستور ساخت جدول را بزنید :

```
CREATE TABLE A (ID INT)
```

```
Messages
NOT ALLOW CREATE TABLE
Msg 3609, Level 16, State 2, Line 1
The transaction ended in the trigger. The batch has been aborted.
```

- **تریگر های سطح سرور**

به تغییرات روی سرور ، عکس العمل نشان می دهد. بر همین اساس دو نوع تریگر است : تریگر هایی که هنگام ورود به سیستم رها می شوند و تریگرهایی که با دستورات DDL رها می شوند.