



دانشگاه صنعتی شریف



دانشگاههای فنی کامپیووتر

طراحی پایگاه داده‌ها

(فصل چهارم: مقدمات پایه سازی و SQL)

مهندی دادبخش

mahdi.dadbakhsh@sharif.edu

۴۰۳۸۴: شماره درس

یکشنبه - سه‌شنبه (۱۶:۳۰ الی ۱۸:۰۰)

۱۴۰۱ - ۱۴۰۲

آشنایی با اسکریپت نویسی

(Error Handling)

(Transaction)

(View)

(Function)

(Stored Procedure)

(Trigger)

پایان

زبان پرس و جوی ساخت یافته (SQL)

آشنایی با محیط SQL Server

انواع داده‌ها در SQL

دستورات تعریف داده‌ها (DDL)

دستورات کار با داده‌ها (DML)

دستورات کنترل داده‌ها (DCL)

بازیابی اطلاعات (Select)

زبان پرس‌وجوی ساخت‌یافته (SQL)

برای پیاده‌سازی طراحی منطقی انجام شده در یک سیستم مدیریت پایگاه داده‌ها نیاز به یک زبان پایگاهی داریم.
زبان SQL (زبان پرس‌وجوی ساخت‌یافته)، زبان استاندارد انجام عملیات پایگاهی در پایگاه‌های داده رابطه‌ای (از دیدگاه کاربردی: جدولی) است.

دستورات زبان SQL به سه دسته کلی تقسیم می‌شوند:

• زبان تعریف داده (DDL – Data Definition Language) :

□ این دستورات برای ایجاد، ویرایش و حذف ساختارهای مورد نیاز بکار می‌روند: Create , Alter , Drop .

• زبان دستکاری داده‌ها (DML – Data Manipulation Language) :

□ این دستورات برای کار با داده‌ها و عملیاتی نظیر درج، ویرایش، حذف و همچنین بازیابی داده‌ها استفاده می‌شوند: Insert , Update , Delete , Select .

• زبان کنترل داده‌ها (DCL – Data Control Language) :

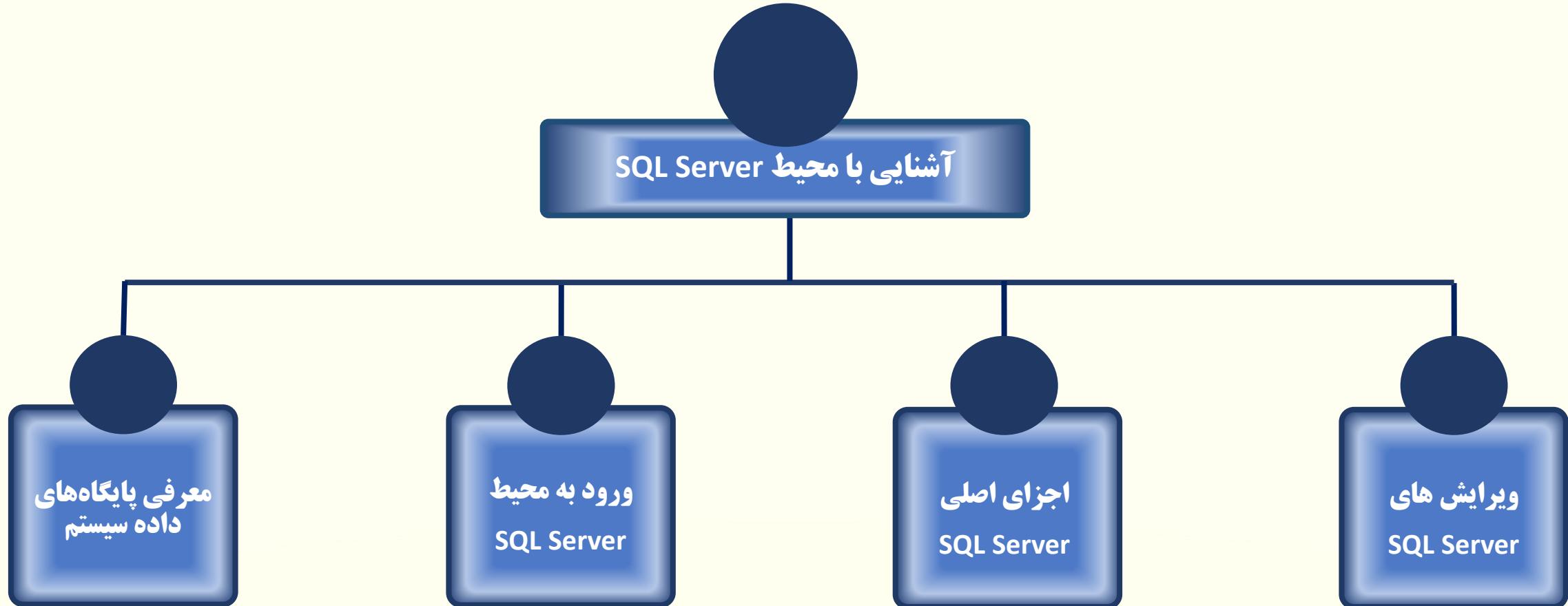
□ این دستورات برای کنترل دسترسی کاربران به پایگاه داده و داده‌ها استفاده می‌شوند: Grant , Deny , Revoke .

برای انجام مکرر عملیات و دسترسی به دستورات SQL از طریق محیط برنامه نویسی می‌توان دستورات را در قالب یکسری ساختارها قرار داد. این ساختارها عبارتند از:

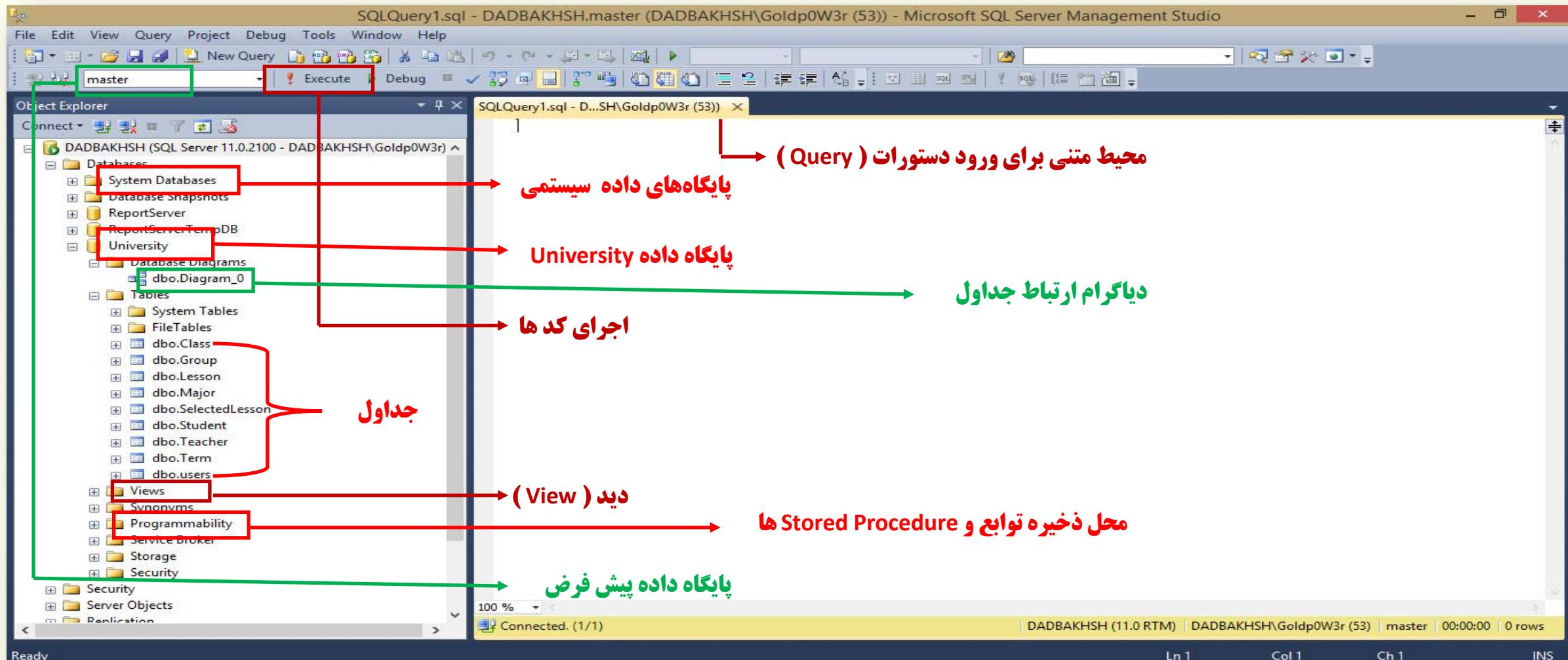
- View
- Function
- Stored Procedure
- Trigger

با ایجاد تراکنش و اجرای دستورات در قالب تراکنش‌ها می‌توان جامعیت داده‌ها را تضمین کرد.

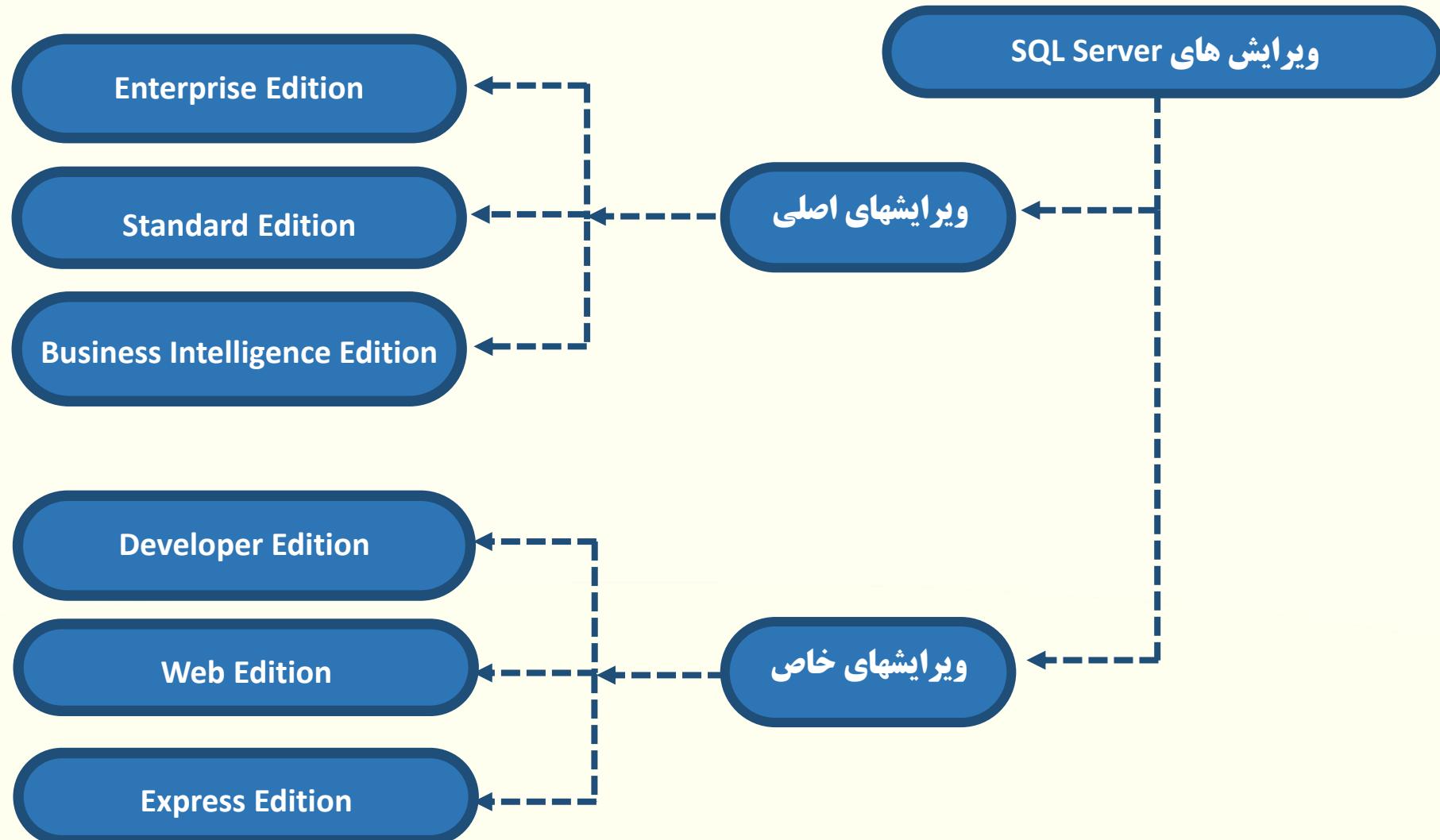




آشنایی با محیط SQL Server



ویرایش های SQL Server



اجزای اصلی SQL Server

۱ - سرویس ها (Services)

سرویس ها برنامه هایی هستند که در پس زمینه اجرا می شوند و رابط کاربر (User Interface) ندارند.
وظیفه سرویس ها ارائه خدمات می باشد.

در واقع سرویس ها عملیات اصلی پایگاه داده را انجام می دهند.

SQL Server دارای ۱۰ سرویس می باشد که می توانید اسمای آنها را در مسیر زیر مشاهده کنید :

Control Panel ---> Administrative Tools ---> Services

۲ - ابزار ها (Tools)

برنامه هایی هستند که قابل رویت هستند.

تمام مواردی که در زیر مجموعه Start Menu در SQL Server مشاهده می شود، ابزارهای SQL هستند.

برخی از این ابزارها عبارتند از :

1 – SQL Data Tools

2 – Integration Services

3 – SQL Server Management Studio

Start Menu ---> SQL Server ---> Document

۳ - مستندات (Documentation)

مستندات به منظور راهنمایی کار با SQL ارائه شده است.



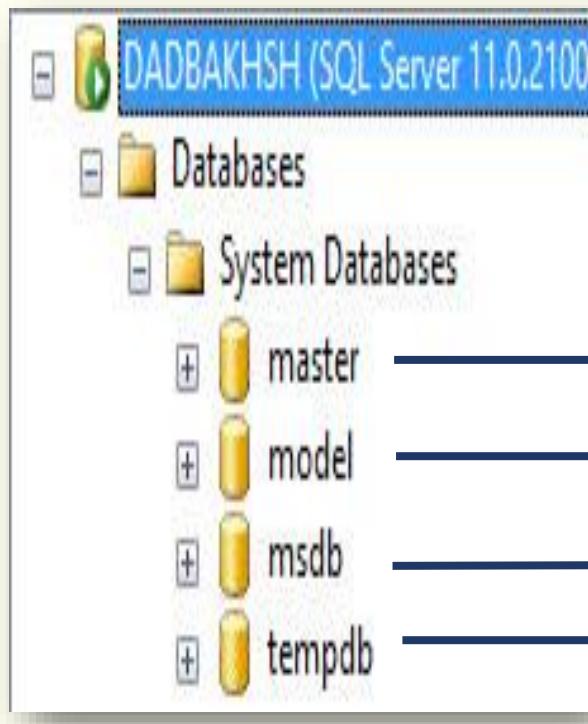
روی SQL Server Management Studio کلیک می کنیم تا صفحه زیر ظاهر شود :



در این قسمت باید مواردی را تنظیم کرد :

- ۱ - ابتدا باید سرویس مورد نظر (Database Engine) را انتخاب کنیم.
- ۲ - سپس باید Server مورد نظر را انتخاب کنیم که معمولاً همان نام کامپیوتر می باشد.
- ۳ - حال باید رمز را وارد کنیم. برای اجتناب از این امر می توان با گزینه Windows Authentication وارد شد.

معرفی پایگاه‌های داده سیستم



۱ - پایگاه داده **master** :

تمام تنظیمات اصلی محیط SQL Server در این پایگاه داده نگهداری می شود.

۲ - پایگاه داده **model** :

مدلی است برای ایجاد پایگاه‌های داده دیگر.

۳ - پایگاه داده **msdb** :

برخی عملیات خودکار سیستم (نظری Backup) در این پایگاه داده نگهداری می شوند.

۴ - پایگاه داده **tempdb** :

عملیات موقت مربوط به SQL در این پایگاه نگهداری می شوند.

انواع داده‌ها در SQL

۱ - نوع داده حرفی :

این نوع داده برای رشته‌های حرفی و عددی مورد استفاده قرار می‌گیرد و انواع مختلفی دارد :

char , nchar , varchar , nvarchar , text , ntext , varchar(ax) , nvarchar(max)

نوع‌های داده‌ای char , varchar و text را بر اساس استاندارد کد اسکی ذخیره می‌کنند (مناسب برای زبان انگلیسی و رشته‌های عددی) .

نوع‌های داده‌ای nvarchar(max) و ntext ، nvarchar ، nchar و nvarchar(max) داده‌ها را بر اساس استاندارد Unicode ذخیره می‌کنند (مناسب برای زبان‌های غیر انگلیسی) .

نوع‌های داده‌ای char و nchar برای داده‌های با طول ثابت بکار می‌روند (مانند شماره دانشجویی و کد ملی) .

نوع‌های داده‌ای varchar و nvarchar برای داده‌های با طول متغیر استفاده می‌شوند (مانند نام، نام خانوادگی، آدرس و ایمیل) .

نکته : داده‌های عددی را که پردازه ریاضی روی آنها انجام نمی‌شود، بنابر دلایلی بهتر است از نوع رشته تعریف شوند. (به نظر شما چه دلایلی ؟)

۲ - نوع داده عددی :

داده‌های عددی در دو نوع صحیح و اعشاری مطرح می‌شوند :

الف) داده‌های صحیح :

ب) داده‌های اعشاری :

۳ - نوع داده تاریخ و زمان :

برای ذخیره سازی تاریخ، زمان و یا ترکیب آنها نوع‌های داده‌ای وجود دارد :

Date , Time , SmallDateTime , DateTime , ...

۴ - نوع داده دودویی :

نوع داده bit : داده‌هایی که مقدار آنها دو حالت دارد، مانند جنسیت.

نوع داده image : برای ذخیره فایل‌های تصویر استفاده می‌شود که البته منسخه شده است.

نوع داده varbinary : برای ذخیره سازی هر نوع فایل باینری (مانند عکس، فیلم و ...) استفاده می‌شود.

۵ - سایر انواع داده :

مالی (Money و SmallMoney) ، جغرافیایی (Geography و Geometry) ، نوع‌های داده‌ای خاص (XML ، Hierarchical و ...)



دستورات تعریف داده‌ها (DDL)

این دستورات برای ایجاد، ویرایش و یا حذف کلیه ساختارهای موجود در پایگاه داده استفاده می‌شود.

این ساختارها عبارتند از :

- پایگاه داده (database)
- جدول (table)
- دید (view)
- تابع (function)
- روال ذخیره شده (stored procedure)
- تریگر (trigger)
- وغیره .

حذف پایگاه داده

ویرایش پایگاه داده

ایجاد پایگاه داده

ایجاد و حذف شیمای پایگاه داده

حذف جدول

ویرایش جدول

ایجاد جدول

نکته مهم در مورد **recreate** شدن جدول



برای ایجاد پایگاه داده در sql server دو روش وجود دارد :

- روش اول : استفاده از حالت wizard
- روش دوم : کدنویسی

هر پایگاه داده در SQL Server حاوی حداقل یک فایل داده و یک فایل سابقه عملیات می‌باشد :

- فایل **data** : تمام داده‌ها، جداول و اطلاعات مورد نیاز در فایل داده ذخیره می‌شود (پسوند اولین فایل داده mdf و سایر فایل‌های داده ndf است) .
- فایل **log** : سابقه عملیات کاربران بر روی پایگاه داده را نگهداری می‌کند (پسوند فایل‌های log برابر ldf می‌باشد) .

فایل‌های سابقه عملیات، قابلیت گروه بندی ندارند ولی فایل‌های داده را می‌توان گروه بندی کرد. اولین فایل داده به طور پیش‌فرض در گروه primary قرار می‌گیرد و قابل تغییر نیست.

اگر گروهی حذف شود، تمام فایل‌های متعلق به آن نیز به‌طور اتوماتیک حذف خواهند شد.

هر فایل داده یا سابقه عملیات اجزایی دارد و به تنظیماتی نیاز دارد :

- نام منطقی (logical name) : نامی است که وجود خارجی ندارد و فقط در کدنویسی برای دسترسی به فایل استفاده می‌شود.
- نام فیزیکی (physical name) : نام واقعی فایل است، بنابراین باید پسوند نیز داشته باشد.
- مسیر فایل (path) : محل ذخیره فایل روی حافظه جانبی (دیسک) را مشخص می‌کند.
- اندازه اولیه فایل (initial size) : اندازه اولیه فایل را مشخص می‌کند. اندازه پیش‌فرض برای فایل داده 5MB و برای فایل سابقه عملیت 1MB است.
- میزان رشد (file growth) : میزان افزایش اندازه فایل را مشخص می‌کند.
- حداکثر اندازه (max size) : حداکثر اندازه فایل را تعیین می‌کند و اگر نامحدود (unlimited) باشد، فایل می‌تواند تا جایی که حافظه موجود باشد رشد کند.
- گروه فایل (file group) : گروهی که فایل به آن تعلق دارد را مشخص می‌کند. این آیتم برای گروه بندی فایل‌ها مورد استفاده قرار می‌گیرد.

روش کدنویسی

روش wizard



ایجاد پایگاه داده - روش wizard

در این روش، برای ایجاد پایگاه داده به ترتیب زیر عمل می‌کنیم :

Database ==> Right Click ==> New Database ==> Enter database name ==> Press OK

با این کار پایگاه داده مورد نظر با یک فایل داده و یک فایل سابقه عملیات ایجاد می‌شود. در این حالت، برای تمامی ویژگی‌های مربوط به پایگاه داده، فایل‌های آن و محل ذخیره فایل‌ها، مقادیر پیش‌فرض اعمال خواهد شد.

در صورت تمایل می‌توانیم تنظیمات مربوط به فایل‌های داده و log و ویژگی‌های آنها را خودمان انجام دهیم :

- در قسمت General تنظیمات فایل داده و log انجام می‌شود.
- در قسمت Option تنظیمات زبان و همچنین نحوه تهیه نسخه پشتیبان انجام می‌شود.
- قسمت Filegroups برای ایجاد گروه‌ها مورد استفاده قرار می‌گیرد.

ایجاد پایگاه داده - روش گدنویسی

برای این منظور از دستور `create database` استفاده می‌کنیم.

حالت اول : در ساده‌ترین شکل فقط با ذکر نام پایگاه داده آن را با تنظیمات پیش‌فرض ایجاد می‌کنیم:

`CREATE DATABASE database_name`

Example : `create database university`

حالت دوم : ایجاد پایگاه داده با تنظیمات دلخواه :

`USE master`

`GO`

`CREATE DATABASE university`

`ON PRIMARY`

`(NAME = 'university_data' , FILENAME = 'd:\database\university_data.mdf' , SIZE = 10MB , MAXSIZE = unlimited , FILEGROWTH = 5MB) ,
FILEGROUP [A]`

`(NAME = 'universityA_data' , FILENAME = 'd:\database\universityA_data.ndf' , SIZE = 5MB , MAXSIZE = 200MB , FILEGROWTH = 5MB) ,
FILEGROUP [B]`

`(NAME = 'universityB_data' , FILENAME = 'd:\database\universityB_data.ndf' , SIZE = 20MB , MAXSIZE = unlimited , FILEGROWTH = 10MB) ,
LOG ON`

`(NAME = 'university_log' , FILENAME = 'd:\database\university_log.ldf' , SIZE = 5MB , MAXSIZE = unlimited , FILEGROWTH = 1MB)`

در این مثال، پایگاه داده University با سه فایل داده با تنظیمات مختلف در سه گروه Primary ، A و B و همچنین یک فایل log ایجاد شده است.

سایر تنظیمات



ایجاد پایگاه داده - سایر تنظیمات

برای انجام تنظیمات بیشتر روی پایگاه داده مورد نظر (University) به صورت مقابل عمل می‌کنیم :
در این قسمت تنظیمات زیر را می‌توان انجام داد :

۱ - تطبیق حروف (Collation) :

- Bin : در این حالت، مقایسه حروف بر حسب کد اسکی انجام می‌شود و به ازای هر حرف یک بایت فضا اختصاص داده می‌شود. بنابراین ۲۵۶ حالت را پشتیبانی می‌کند و برای حروف فارسی استانداردی در نظر گرفته نشده است.
- Bin2 : در این حالت، مقایسه حروف بر حسب unicode انجام می‌شود و به ازای هر حرف دو بایت فضا اختصاص داده می‌شود. بنابراین ۶۵۵۳۶ حالت را پشتیبانی می‌کند و از حروف فارسی نیز پشتیبانی می‌شود.
- Case (In) Sensitive : در این حالت، حساسیت بین حروف بزرگ و کوچک مطرح می‌شود (برای زبان فارسی کاربردی ندارد ولی برای زبان انگلیسی بکار می‌رود).
- Accent (In) Sensitive : در این حالت، حساسیت نسبت به تلفظ حروف مطرح است (برای زبان انگلیسی و فارسی کاربردی ندارد).
- Kana (In) Sensitive : در این حالت، حساسیت نسبت به نحوه نگارش حروف مطرح می‌شود (برای زبان انگلیسی و فارسی کاربردی ندارد).
- Width (In) Sensitive : در این حالت، حساسیت نسبت به پهنهای حروف مطرح است (برای زبان انگلیسی و فارسی کاربردی ندارد).

با توجه به توضیحات فوق، گزینه مناسب برای زبان فارسی Persian.100_CI_AI می‌باشد.

۲ - سازگاری با نسخه‌های قبلی (Compatibility Level) :

اگر بخواهیم پایگاه داده که می‌سازیم با یک نسخه خاصی از sql server سازگار باشد، آن نسخه را انتخاب می‌کنیم.

۳ - مدل بازیابی (Recovery Model) :

- Full : همه عملیات کاربر ثبت می‌شود (بدون حذف از فایل). در این حالت، حجم فایل log بسیار زیاد خواهد بود.
- Bulk-logged : برخی از عملیات کاربر ثبت می‌شود (بدون حذف از فایل). در این حالت، حجم فایل کمتر از حالت قبل است. اینکه چه عملیاتی ثبت شود به عهده سیستم است و developer کنترلی روی آن ندارد.
- Simple : در این حالت نیز برخی از عملیات کاربر ثبت می‌شود، با این تفاوت که امکان overwrite کردن داده‌ها وجود دارد. در این حالت، مدت زمان تعیین می‌شود و پس از سپری شده این زمان اطلاعات در فایل log بازنویسی می‌شود. این بازه زمانی را recovery interval می‌گویند و از طریق زیر تنظیم می‌شود :

Server ==> Right Click ==> Properties ==> Database Setting ==> Recovery Interval

ویرایش پایگاه داده

روش اول : استفاده از حالت **wizard** :

- برای این منظور روی پایگاه داده مورد نظر کلیک راست کرده گزینه **properties** را اجرا می کنیم :
- در قسمت **file** می توان تنظیمات فایل های داده و **log** را ویرایش کرد.
- در قسمت **file group** می توان تنظیمات گروه های فایلی را ویرایش کرد.
- در قسمت **option** نیز مواردی نظیر **collation** و ... را می توان ویرایش نمود.

روش دوم : استفاده از دستور **alter database** :

ALTER DATABASE *database_name*

- ویرایش هایی که می توان با **alter** انجام داد به چند حالت خلاصه می شود :
 - اضافه کردن فایل داده یا **log** جدید :
 - حذف فایل داده یا **log** موجود :
 - ویرایش فایل داده یا **log** :

ADD FILE (...)

REMOVE FILE ' *logical_name* '

MODIFY FILE ' *logical_name* ' (...)

- در هر لحظه فقط می توان یکی از این عملیات را به همراه **alter** انجام داد. به طور مثال، برای افزودن فایل داده یا **log** جدید به پایگاه داده **university** داریم :

ALTER DATABASE *university*

ADD FILE

(NAME = ' *university2_data* ' , FILENAME = ' d:\database\university2_data.ndf ')

حذف پایگاه داده

روش اول : استفاده از حالت **wizard** :

- برای این منظور روی پایگاه داده مورد نظر کلیک راست کرده گزینه **delete** را اجرا می کنیم :

روش دوم : استفاده از دستور **drop database** :

DROP DATABASE *database_name*

- با اجرای این دستور پایگاه داده مورد نظر به همراه تمام ساختارها و داده ها از سیستم حذف می شود.
- به طور مثال برای حذف پایگاه داده **university** به صورت زیر عمل می کنیم :

DROP DATABASE *university*

ایجاد و حذف شمای پایگاه داده

- شمای پایگاه داده‌ها معرف یک فضای نام (name space) در یک پایگاه داده است و شامل تعدادی جدول، نوع، دامنه، دید، محدودیت و دیگر انواع اشیاء مرتبط با یک برنامه کاربردی است. برای گروه بندی جداول، دیدها، روال‌ها، توابع و ... در داخل پایگاه داده از شما استفاده می‌شود.

- دستور ایجاد و تعریف شمای پایگاه داده :

CREATE SCHEMA *schema_name*

- دستور حذف شمای پایگاه داد :

DROP SCHEMA *schema_name*

- با تعریف اشیاء در یک شما، نام شما به عنوان پیشوند نام آنها لحاظ می‌شود و از تداخل نام اشیاء برای کاربردهای مختلف جلوگیری می‌شود.

- به‌طور مثال شمای **dbcourse** به صورت زیر ایجاد می‌کند :

CREATE SCHEMA *dbcourse*

(مثال: **dbcourse.student** جدول student از شمای **dbcourse**)



ایجاد جدول

روش اول : استفاده از حالت **wizard** :

- در پایگاه داده مورد نظر روی گزینه Tables کلیک راست کرده، گزینه new table را اجرا می کنیم. حال فیلدها را یک به یک تعریف می نماییم.

روش دوم : کدنویسی :

- برای این منظور ابتدا با استفاده از USE پایگاه داده جاری و فعال را مشخص می کنیم. سپس برای ایجاد جدول از دستور create table استفاده می کنیم :

```
USE database_name  
CREATE TABLE table_name  
(  
    column_name data_type (size) { NULL | NOT NULL | PRIMARY KEY } ,  
    ...  
    FOREIGN KEY (column_name2) REFERENCES table_name2  
)
```

نام ستون (column_name) : در این قسمت برای ستون مورد نظر نامی نسبت می دهیم.

نوع داده (data_type) : نوع داده هر ستون یا فیلد را باید مشخص کنیم.

ضروری / غیرضروری بودن (null / not null) : اگر ستونی به صورت null باشد هنگام ورود اطلاعات کاربر می تواند مقداری برای آن وارد نکند و ستون فاقد مقدار باشد. حال اگر ستون not null باشد ضروری است و حتما باید برای آن مقدار وارد کرد.

کلید اصلی (primary key) : در صورت نیاز به داشتن کلید اصلی، مقابل ستون مورد نظر واژه key را می نویسیم . فیلدی که کلید اصلی باشد به طور اتوماتیک not null نیز خواهد بود.

کلید خارجی (foreign key) : اگر این جدول با جدول دیگری (table_name2) که column_name2 کلید اصلی آن است، ارتباط داشته باشد، باید کلید خارجی جدول را مشخص کرد. کلید خارجی باید با column_name2 هم نوع و هم اندازه باشد.

مثال

خصوصیات ستون



ایجاد جدول - خصوصیات ستون

هنگام ایجاد جدول می‌توان در مورد هر ستون خصوصیاتی را تنظیم کرد :

۱ - name : نام ستون را مشخص می‌کند.

۲ - allow null : این گزینه بیانگر این است که ستون می‌تواند خالی باشد (به عبارت دیگر، ستون غیرضروری است).

۳ - data type : نوع داده ستون را مشخص می‌کند.

۴ - default value or binding : مقدار پیش فرض ستون را مشخص می‌کند.

۵ - length : طول نوع داده ستون را مشخص می‌کند.

۶ - collation : برای تطبیق حروف بکار می‌رود و فقط روی داده‌های حرفی معنی دار است.

۷ - computed column specification : برای تنظیم ستون محاسباتی بکار می‌رود. ستون محاسباتی ستونی است که مقدارش به طور اتوماتیک از روی ستون‌های

دیگر جدول محاسبه می‌شود. ستون محاسباتی از نوع فقط خواندنی است و برای آن نوع در نظر گرفته نمی‌شود. دو حالت داریم :

الف) اگر is persisted برابر yes باشد، نتایج محاسبه ذخیره می‌شود.

ب) اگر is persisted برابر no باشد، نتایج محاسبه ذخیره نمی‌شود.

۸ - description : شرحی را می‌توان در مورد ستون مورد نظر اعمال کرد.

۹ - identity specification : یک ستون عددی است که به صورت auto number عمل می‌کند، یعنی مقدار آن به طور اتوماتیک توسط سیستم تولید می‌شود. این

ستون نیز از نوع فقط خواندنی است و کاربر نمی‌تواند مقداری برای آن وارد کرده و یا مقدار آن را ویرایش کند.

هنگام تعریف این نوع ستون باید دو پارامتر مقداردهی شود :

الف) identity seed : مقدار اولیه برای شروع شمارش

ب) identity increment : میزان افزایش در هر بار شمارش



ایجاد جدول - مثال

```
□ Use university
    -- ایجاد جدول رشته تحصیلی
□ create table Major
(
    Mcode char(3) primary key,      -- کد رشته تحصیلی
    Mname nvarchar(30) not null,   -- نام رشته تحصیلی
    StCount smallint null         -- تعداد دانشجو
)
```

```
□ Use university
    -- ایجاد جدول درس
□ create table Lesson
(
    Lcode char(6) primary key,      -- کد درس
    Lname nvarchar(30) not null,   -- نام درس
    Unit tinyint not null,         -- تعداد واحد
    PreLcode char(6) null,          -- کد درس پیش نیاز
    Mcode char(3) not null,          -- کد رشته تحصیلی
    Foreign Key (Mcode) References Major
)
```



ویرایش جدول

روش اول : استفاده از حالت **wizard** :

- در پایگاه داده مورد نظر روی جدولی که می خواهیم ویرایش کنیم کلیک راست کرده، گزینه **design** را اجرا می کنیم و تغییرات را اعمال می کنیم.

روش دوم : کدنویسی :

- برای این منظور ابتدا با استفاده از **USE** پایگاه داده جاری و فعال را مشخص می کنیم. سپس برای ویرایش جدول از دستور **alter table** استفاده می کنیم :

USE *database_name*

ALTER TABLE *table_name*

ADD *new_column_name data_type (size)*

DROP COLUMN *column_name*

ALTER COLUMN *column_name*

- با هر بار اجرای دستور **alter** می توان یکی از تغییرات زیر را اعمال نمود :

- افزودن ستون جدید

- حذف ستون موجود

- ویرایش ستون موجود

USE *university*

ALTER TABLE *lesson*

ADD *lesson_type nvarchar(20)*

مثال : اضافه کردن ستونی "نوع درس" به جدول "درس"



حذف جدول

روش اول : استفاده از حالت **wizard** :

- در پایگاه داده مورد نظر روی جدولی که می خواهیم ویرایش کنیم کلیک راست کرده، گزینه **delete** را اجرا می کنیم.

روش دوم : کدنویسی :

- برای این منظور ابتدا با استفاده از **USE** پایگاه داده جاری و فعال را مشخص می کنیم. سپس برای ویرایش جدول از دستور **drop table** استفاده می کنیم :

USE *database_name*

DROP TABLE *table_name*

مثال : حذف جدول "درس"

USE *university*

DROP TABLE *lesson*

نکته : اگر جدول یا جداولی با جدولی که می خواهیم حذف کنیم در رابطه بوده و به آن وابسته باشند، هنگام حذف جدول ممکن است سیستم بسته به نوع ارتباط جداول، اجازه انجام این کار را به ما ندهد.

نکته مهم در مورد `recreate` شدن جدول

برخی عملیات روی جداول منجر به `recreate` شدن جدول می‌شوند. به طور مثال، اگر بخواهیم ستونی را بین دو ستون جدول `student` اضافه کنیم، مراحل زیر انجام می‌شود:

- حذف تمام محدودیت‌ها از جدول `student`.
- ایجاد جدولی به نام `temp` با تمامی فیلدهای جدول `student`.
- اعمال محدودیت‌های جدول `student` روی جدول `temp`.
- انتقال داده‌های جدول `student` به جدول `temp`.
- حذف جدول `student`.
- تغییر نام جدول `temp` به `student`.

سیستم به طور پیش فرض اجازه چنین تغییراتی را نمی‌دهد و از آن جلوگیری می‌کند.
برای اینکه در فرایند طراحی پایگاه داده دچار مشکل نشوید ابتدا ، این امکان را برای خود فعال کنید.
برای این منظور از مسیر زیر تیک گزینه "... prevent saving change" را بردارید:

Tools ==> Option ==> Designers ==> Prevent saving change ...

دستورات کار با داده‌ها (DML)

این دستورات برای کار با داده‌ها، دستکاری داده‌ها و بازیابی آنها مورد استفاده قرار می‌گیرد .
این دستورات عبارتند از :

- درج داده در جدول (insert)
- ویرایش داده‌های جدول (update)
- حذف داده‌های جدول (delete)
- بازیابی اطلاعات (select)

حذف داده‌ها

ویرایش داده‌ها

درج داده در جدول



درج داده در جدول

برای درج داده در جدول از دستور **insert** استفاده می‌شود. این دستور را می‌توان در چهار حالت مختلف بکار برد :

- **حالت اول :** فیلدهای جدول را به طور مستقیم مقداردهی کنیم :

```
USE database_name  
INSERT INTO table_name  
[ ( column01 , column02 , ... ) ]  
VALUES ( value01 , value02 , ... )
```

نکته ۱ : نوشتن نام ستون‌ها الزامی نیست مگر اینکه بخواهیم ترتیب ورود داده‌ها را تغییر دهیم و یا اینکه بخواهیم برخی فیلدها را مقدار دهی کنیم.

نکته ۲ : اگر نام ستونی قید نشود، ابتدا بررسی می‌شود که آیا مقدار پیش فرض دارد یا خیر. اگر فیلد مقدار پیش فرض داشته باشد با آن پر می‌شود و اگر مقدار پیش فرض نداشته باشد، بررسی می‌شود که آیا ستون **allow null** است یا **not null**. اگر باشد مقدار **null** در آن قرار می‌گیرد و درغیراين صورت خطأ رخ می‌دهد.

- **حالت دوم :** در این حالت، ستون‌های جدول با مقادیر پیش فرض مقداردهی می‌شوند:

```
INSERT INTO table_name DEFAULT VALUES
```

نکته : این دستور به شرطی درست عمل می‌کند که تمامی فیلدها دارای مقدار پیش فرض باشند.

- **حالت سوم :** در این حالت، مقادیر مورد نیاز برای درج در جدول از دستور **select** دریافت می‌شود :

```
INSERT INTO table_name  
( column01 , column02 , ... )  
SELECT ...
```

نکته : در این حالت، باید خروجی **select** از نظر نوع، تعداد و ترتیب فیلدها با ستون‌های موجود در **insert** متناسب باشد.

- **حالت چهارم :** در این حالت، مقادی مورد نیاز جهت درج در جدول با اجرای یک روال (**procedure**) بدست می‌آید :

```
INSERT INTO table_name  
( column01 , column02 , ... )  
EXECUTE procedure_name
```

نکته : در این حالت، باید خروجی **procedure** از نظر نوع، تعداد و ترتیب فیلدها با ستون‌های موجود در **insert** متناسب باشد.

مثال



درج داده در جدول - مثال

```
use University
Insert Into Major
(Mcode,Mname,StCount)
Values('100','فناوري اطلاعات',0),
      ('200','نرم افزار',0),
      ('300','سخت افزار',0),
      ('400','گرافيك',0),
      ('500','معماري',0),
      ('600','روابط عمومي',0)
```

	Mcode	Mname	StCount
▶	100	فناوري اطلاعات	0
	200	نرم افزار	0
	300	سخت افزار	0
	400	گرافيك	0
	500	معماري	0
	600	روابط عمومي	0

ویرایش داده‌های جدول

برای ویرایش و اصلاح داده‌های موجود در جدول از دستور update استفاده می‌شود :

```
USE database_name  
UPDATE table_name  
SET column_name01 = value01 , ...  
[WHERE condition01 , condition02 , ... ]
```

نکته ۱ : اگر قسمت where را نداشته باشیم و شرطی را اعمال نکنیم، تغییرات روی تمام رکوردهای جدول صورت خواهد گرفت.

مثال : یک واحد به تعداد دانشجویان رشته "فناوری اطلاعات" اضافه می‌کنیم :

```
USE university  
UPDATE major  
SET stcount = stcount + 1  
WHERE mname = 'فناوری اطلاعات'
```

نکته ۲ : می‌توان رکوردهای تغییریافته را مشاهد کرد. این کار با استفاده از output بعد از set و قبل از where قرار می‌گیرد :

OUTPUT inserted . column_name ==> مقادیر فیلد پس از ویرایش

OUTPUT deleted . column_name ==> مقادیر فیلد پیش از ویرایش

نکته ۳ : ممکن است شرط موجود در where مربوط به جدول دیگری باشد. در این صورت، دو روش وجود دارد :

- روش اول : نوشتن where مقابل sub query
- روش دوم : استفاده از پیوند جداول (join)

نکته ۴ : می‌توان در قسمت set نیز از sub query استفاده کرد.

مثال



ویرایش داده‌های جدول – مثال

▪ مثال : تعداد دانشجویان رشته‌هایی که تعداد دانشجوی آنها صفر است یک واحد اضافه کند.

```
USE university  
UPDATE major  
SET stcount = stcount + 1  
OUTPUT inserted . mname , deleted . Stcount as 'Before' , inserted . Stcount as 'After'  
WHERE stcount = 0
```

▪ مثال : قیمت کالاهایی که در نام مدل آنها کلمه Bike وجود دارد، ۱۰ درصد افزایش یابد.

```
UPDATE product  
SET price = price * 1.1  
OUTPUT inserted . name , deleted . price as 'Before' , inserted . price as 'After'
```

▪ روشن اول : نوشت sub query مقابل where :

```
UPDATE product  
SET price = price * 1.1  
FROM product as P  
INNER JOIN productmodel as PM  
ON P . productmodelID = PM . productmodelID  
WHERE PM . name LIKE '%Bike%'
```

▪ روشن دوم : استفاده از join :



حذف داده‌ها از جدول

- برای حذف رکوردهای جدول می‌توان از دستور `delete` استفاده کرد :

```
USE database_name
DELETE FROM table_name
[WHERE condition01, condition02, ... ]
```

- نکته ۱ : اگر شرطی اعمال نکنیم تمام سطرهای جدول حذف می‌شود.

```
OUTPUT deleted . column_name
```

- نکته ۲ : برای مشاهده رکوردهای حذف شده می‌توان از `output` استفاده کرد :

- نکته ۳ : در صورت نیاز می‌توان همانند `update` ، در `delete` نیز از `join` و `sub query` استفاده کرد.

دستورات کنترل داده‌ها (DCL)

صفت



برای بازیابی اطلاعات از جداول پایگاه داده، از دستور select استفاده می‌شود.

فرم کلی دستور select به صورت زیر است:

```
SELECT <clause>
FROM <clause>
WHERE <clause>
GROUP BY <clause>
HAVING <clause>
ORDER BY <clause>
```

به جز قسمت select بقیه موارد فوق اختیاری بوده و فقط در صورت نیاز از آنها استفاده می‌شود.

نکته: ترتیب قرار گرفتن آیتم‌های فوق اهمیت بسیاری دارد و هرگز نباید جابجایی در آنها صورت گیرد.

Select

From

Where

Aggregation

Group by

Having

Order by

Union , Intersect , Except

Nested Select



پاز پایی اطلاعات – پخش Select – قسمت اول

۱ - نام ستون ها :

نام ستون هایی را که می خواهیم در خروجی نمایش داده شود، مقابل select می نویسیم :

```
SELECT mcode , mname FROM major
```

اگر به جای نام ستون ها علامت ستاره (*) قرار دهیم تمام ستون های جدول نمایش داده خواهد شد :

```
SELECT * FROM major
```

۲ - توابع :

می توان از توابع موجود در sql نیز در مقابل select استفاده کرد.

```
SELECT FirstName , LastName , YEAR ( Birthdate ) as 'Year' FROM student
```

برای مشاهده لیست توابع موجود در سیستم مسیر زیر را دنبال می کنیم :

Programmability ==> functions ==> System Functions

۳ - نام مستعار (Alias) :

نام مستعار نامی است که می توان برای ستون های نشان داده شده توسط دستور select در نظر گرفت. این کار برای دو منظور انجام می شود :

الف) در نظر گرفتن عنوان برای فیلدی از جدول :

```
SELECT mcode as 'نام رشته' FROM major
```

ب) قرار دادن عنوان برای ستون های فاقد نام :

```
SELECT FirstName , LastName , YEAR ( Birthdate ) as 'سال تولد' FROM student
```

نکته: نوشتن نام مستعار برای ستون های فاقد نام الزامی است.

۴ - عبارت محاسباتی :

می توان مقابل select از عبارات محاسباتی نیز استفاده کرد :

```
SELECT FirstName + LastName as 'FullName' FROM student
```

بازیابی اطلاعات – بخش Select – قسمت دوم

۵ - حذف سطرهای تکراری (**Distinct**) :
هنگام نمایش خروجی select ممکن است سطر یا سطرهای تکراری وجود داشته باشد. با نوشتن **distinct** بعد از **select** می‌توان از نمایش سطرهای تکراری جلوگیری کرد :

`SELECT DISTINCT FirstName , LastName FROM student`

۶ - مشاهده تعدادی رکورد از بالای جدول :
اگر بخواهیم تعدادی رکورد از بالای جدول را مشاهده کنیم، از **Top** استفاده می‌کنیم. **Top** به سه صورت قابل استفاده است :
الف) نمایش **n** رکورد از بالای جدول (**Top n**) :

`SELECT TOP 10 * FROM student`

ب) نمایش **n** درصد از رکوردهای بالای جدول (**Top n Percent**) :

`SELECT TOP 10 PERCENT * FROM student`

ج) نمایش **n** درصد از رکوردهای بالای جدول با نمایش موارد مشابه (**Top n with ties**) :
فرض کنید نام و سن افراد به شرح زیر است : علی (۲۵)، حسن (۲۵)، حسین (۲۵)، رضا (۲۰) و محمد (۲۰)
حال می‌خواهیم ببینیم چه کسی بالاترین سن را دارد. به دو صورت می‌توان عمل کرد :
حالت ۱ :

در این حالت فقط علی نمایش داده می‌شود، در حالی که حسن و حسین نیز ۲۵ سال سن دارند.

`SELECT TOP 1 FirstName , Age FROM student ORDER By Age DESC`

حالت ۲ :

`SELECT TOP 1 with Ties FirstName , Age FROM student ORDER By Age DESC`

در این حالت هر سه نفر که ۲۵ سال سن دارند نمایش داده می‌شوند.
نکته: **Top** به تنها ی مهندی دار نیست و معمولاً به همراه **Order by** بکار می‌رود.

پازیابی اطلاعات – بخش Select – قسمت سوم

۷ - هدایت خروجی به یک جدول جدید :

می‌توان خروجی حاصل از select را در یک جدول جدید ذخیره کرد :

```
SELECT column_name , ... INTO new_table_name FROM table_name
```

نکته: ستون‌ها و سطرهای جدول جدید برابر ستون‌ها و سطرهایی است که در خروجی Select ظاهر می‌شوند.

۸ - ذخیره در جدول موقت :

می‌توان خروجی حاصل از select را در یک جدول موقت ذخیره کرد. برای این منظور دو حالت وجود دارد :

الف) جدول موقت محلی (local) :

این نوع جدول مختص یک connection یا کاربر خاص است یعنی کاربران دیگر می‌توانند جدول موقت محلی با همین نام داشته باشند و هر کاربر به جدول مربوط به خود دسترسی دارد. برای ایجاد چنین جدولی قبل از نام جدول جدید از علامت # استفاده می‌کنیم :

```
SELECT column_name , ... INTO #new_table_name FROM table_name
```

نکته: این جدول به محض قطع شدن اتصال کاربر از بین می‌رود.

ب) جدول موقت سراسری (global) :

این نوع جدول مختص یک connection یا کاربر خاص نیست، یعنی وقتی کاربری جدولی با نام test ایجاد می‌کند کاربران دیگر به آن دسترسی دارند و دیگر نمی‌توانند جدولی با این نام بسازند. برای ایجاد چنین جدولی قبل از نام جدول جدید علامت ## قرار می‌دهیم :

```
SELECT column_name , ... INTO ##new_table_name FROM table_name
```

نکته: این جدول زمانی از بین می‌رود که اتصال آخرین کاربر قطع شود.

نکته: جداول موقت در قسمت Temporary Table ذخیره می‌شوند.

پازیابی اطلاعات – بخش From – قسمت اول

۱ - نام جدول :

نام جدول یا جداولی را که می‌خواهیم داده‌های آنها را نمایش دهیم مقابل `from` می‌نویسیم :

```
SELECT mcode , mname FROM major
```

۲ - توابع جدولی :

مقابل `from` می‌توان نام توابع جدولی را نوشت و اطلاعات مورد نظر را از خروجی آن توابع استخراج کرد :

```
SELECT column_name , ... FROM Tabled_Function
```

تابع جدولی تابعی است که خروجی آن یک جدول است.

۳ - دید (view) :

مقابل `from` می‌توان نام `view` را نوشت و اطلاعات مورد نظر را از خروجی آن استخراج کرد.

```
SELECT column_name , ... FROM view_name
```

دید (`view`) یک `query` ذخیره شده است که به منظور خاصی نوشته شده است. برای مشاهده لیست `view` ها مسیر زیر را دنبال کنید :

Database_name ==> Views ==> System Views

نکته: تفاوت `view` و تابع جدولی این است که `view` قادر پارامتر ورودی است.

نکته: برای مشاهد محتویات `view` روی آن کلیک راست می‌کنیم و گزینه `design` را اجرا می‌کنیم.

۴ - sub query :

در مقابل `from` می‌توان از `select` دیگری استفاده کرد. در این حالت فرمت `select` به صورت زیر خواهد بود :

```
SELECT column_name , ... FROM ( SELECT column_name , ... ) as virtual_table_name
```

درواقع خروجی `select` داخلی به عنوان یک جدول مجازی در نظر گرفته می‌شود.

مثال sub query

بازیابی اطلاعات – بخش From – مثال sub query

مثال ۱ :

```
SELECT TOP 3 *
FROM ( SELECT FirstName , LastName
      FROM student
     ) as ST
ORDER By LastName
```

در این مثال select داخلي نام و نام خانوادگي دانشجويان را در قالب جدول مجازی St بازمي گرداند و select بيرونی سه نفر اول را برحسب نام خانوادگي نمایش می دهد.

مثال ۲ :

```
SELECT *
FROM ( SELECT TOP 3 FirstName , LastName , Age
      FROM student
     ORDER By Age DESC
    ) as ST
ORDER By LastName
```

در اين مثال select داخلي نام ، نام خانوادگي و سن سه دانشجويي که بيشترین سن را دارند ارائه می دهد و select بيرونی آنها را برحسب نام خانوادگي مرتب کرده نمایش می دهد.

پاز پایی اطلاعات - پخش From - پیوند جداول

Inner Join

Left Outer Join

Right Outer Join

Full Outer Join

Cross Join

Examples

۵ - پیوند جداول (Join) :

از join برای مشاهده اطلاعات از چندین بخش مختلف (چندین جدول یا view) استفاده می شود .
پیوند جداول انواع مختلفی دارد :

- پیوند داخلی (Inner Join)
- پیوند خارجی (Outer join)
- پیوند خارجی چپ (Left Outer Join)
- پیوند خارجی راست (Right Outer Join)
- پیوند خارجی کامل (Full Outer Join)
- پیوند ضربدری (Cross Join)

مثال : فرض کنید دو جدول رشته تحصیلی (major) و درس (lesson) را به صورت زیر داریم :

mcode	mname	stcount
۱۰۰	فناوری اطلاعات	۲۰
۲۰۰	سخت افزار	۰
۳۰۰	معماری	۳۰

Icode	Iname	...	mcode
۱۰۰۰	بانک اطلاعاتی		۱۰۰
۲۰۰۰	طراحی الگوریتم		۱۰۰
۳۰۰۰	نقشه کشی		۳۰۰
۴۰۰۰	تاریخ		Null
۵۰۰۰	فارسی		Null



پازیابی اطلاعات – پخش From – پیوند داخلی

متداول ترین نوع پیوند جداول، پیوند داخلی است و فرم کلی آن به صورت زیر است :

```
SELECT column_name , ...
FROM left_table_name INNER JOIN right_table_name
ON left_table_name . column_name = right_table_name . column_name
```

در این نوع پیوند، اطلاعات مشترک در هر دو جدول سمت چپ و راست نشان داده می شود.

مثال :

```
SELECT major . Mname , lesson . Lname
FROM major INNER JOIN lesson
ON major . mcode = lesson . mcode
```

خروجی به صورت زیر خواهد بود :

mname	Iname
فناوری اطلاعات	بانک اطلاعاتی
فناوری اطلاعات	طراحی الگوریتم
معماری	نقشه کشی

پاز پایی اطلاعات – پیش من - پیوند بیرونی چپ

فرم کلی پیوند بیرونی چپ به صورت زیر است :

```
SELECT column_name , ...
FROM left_table_name LEFT OUTER JOIN right_table_name
ON left_table_name . column_name = right_table_name . column_name
```

در این نوع پیوند، علاوه بر داده‌های مشترک و دارای انطباق در هر دو جدول، داده‌هایی از جدول سمت چپ که هیچ انطباقی با جدول سمت راست ندارند نیز نشان داده می‌شوند.

مثال :

```
SELECT major . Mname , lesson . Lname
FROM major LEFT OUTER JOIN lesson
ON major . mcode = lesson . mcode
```

خروجی به صورت زیر خواهد بود :

mname	lname
فناوری اطلاعات	بانک اطلاعاتی
فناوری اطلاعات	طراحی الگوریتم
معماری	نقشه کشی
سخت افزار	Null

پاز پایی اطلاعات – پیش مند پیرونی راست

فرم کلی پیوند بیرونی راست به صورت زیر است :

```
SELECT column_name , ...
FROM left_table_name RIGHT OUTER JOIN right_table_name
ON left_table_name . column_name = right_table_name . column_name
```

در این نوع پیوند، علاوه بر داده‌های مشترک و دارای انطباق در هر دو جدول، داده‌هایی از جدول سمت راست که هیچ انطباقی با جدول سمت چپ ندارند نیز نشان داده می‌شوند.

مثال :

```
SELECT major . Mname , lesson . Lname
FROM major RIGHT OUTER JOIN lesson
ON major . mcode = lesson . mcode
```

خروجی به صورت زیر خواهد بود :

mname	lname
فناوری اطلاعات	بانک اطلاعاتی
فناوری اطلاعات	طراحی الگوریتم
معماری	نقشه کشی
Null	تاریخ
Null	فارسی

پازیابی اطلاعات – پخش From – پیوند بیرونی کامل

فرم کلی پیوند بیرونی کامل به صورت زیر است :

```
SELECT column_name , ...
FROM left_table_name FULL OUTER JOIN right_table_name
ON left_table_name . column_name = right_table_name . column_name
```

در این نوع پیوند، علاوه بر داده‌های مشترک و دارای انطباق در هر دو جدول، داده‌هایی از جدول سمت راست ندارند و همچنین داده‌هایی از جدول سمت راست که هیچ انطباقی با جدول سمت چپ ندارند نیز نشان داده می‌شوند.

مثال :

```
SELECT major . Mname , lesson . Lname
FROM major FULL OUTER JOIN lesson
ON major . mcode = lesson . mcode
```

خروجی به صورت زیر خواهد بود :

mname	lname
فناوری اطلاعات	بانک اطلاعاتی
فناوری اطلاعات	طراحی الگوریتم
معماری	نقشه کشی
سخت افزار	Null
Null	تاریخ
Null	فارسی

بازیابی اطلاعات – پخش From – پیوند ضربدری

فرم کلی پیوند ضربدری به صورت زیر است :

```
SELECT column_name , ...  
FROM left_table_name CROSS JOIN right_table_name
```

در این نوع پیوند، تمام رکوردهای جدول اول با تمام رکوردهای جدول دوم بدون در نظر گرفتن انطباق بین آنها، به صورت ضرب دکارتی ترکیب می‌شوند.

مثال :

```
SELECT major . Mname , lesson . Lname  
FROM major CROSS JOIN lesson
```

نکته : پیوند ضربدری تنها نوع پیوندی است که بخش `ON` را ندارد.

پازیابی اطلاعات – پخش From – مثالهای پیوند جداویل

مثال ۱ : دستور select ای بنویسید که نام و نام خانوادگی و نام رشته تحصیلی آنها را نمایش دهد.

پاسخ مثال ۱

مثال ۲ : دستور select ای بنویسید که نام درس، تعداد واحد و نام خانوادگی مدرسی که درس را تدریس می‌کند نمایش دهد.

پاسخ مثال ۲

مثال ۳ : دستور select ای بنویسید که نام، نام خانوادگی دانشجو و نام درس او را نمایش دهد.

پاسخ مثال ۳



Major

Student

```
SELECT Student . FirstName , Student . LastName , Major . mname  
FROM Student  
INNER JOIN Major ON Student . mcode = Major . mcode
```

Lesson

Group

Teacher

```
SELECT Lesson . Iname , Lesson . unit , Teacher . LastName  
FROM Lesson  
INNER JOIN [Group] ON Lesson . Icode = [Group] . Icode  
INNER JOIN Teacher ON Teacher . tcode = [Group] . tcode
```



```
SELECT Student.FirstName , Student.LastName , Lesson.Iname  
FROM Student  
INNER JOIN SelectedLesson ON Student.stno = SelectedLesson.Stno  
INNER JOIN [Group] ON [Group].gcode = SelectedLesson.gcode  
INNER JOIN Lesson ON Lesson.Icode = [Group].Icode
```

پازیابی اطلاعات – پخش Where

برای محدود کردن سطراها از where استفاده می شود. این کار با اعمال شرایطی خاص امکان پذیر است. عملگرهایی که می توان در شرط where استفاده کرد، عبارتند از :

۱ - عملگرهای مقایسه‌ای : < ، <= ، > ، >= و <> .

۲ - عملگرهای منطقی : not و or و and .

۳ - عملگرهای خاص : is not null ، is null ، like ، in ، between .

عملگر Between : قرار داشتن بین دو مقدار را بررسی می کند :

```
SELECT * FROM major WHERE stcount BETWEEN 10 AND 100
```

عملگر In : قرار داشتن در مجموعه‌ای از مقادیر را بررسی می کند :

```
SELECT * FROM class WHERE classtype IN ('آزمایشگاه', 'سایت', 'تئوری')
```

عملگر Is Null : رکوردهایی را باز می گرداند که مقدار فیلد مورد نظر آنها null باشد :

```
SELECT * FROM major WHERE stcount IS NULL
```

عملگر Is Not Null : رکوردهایی را باز می گرداند که مقدار فیلد مورد نظر آنها null نباشد :

```
SELECT * FROM major WHERE stcount IS NOT NULL
```

عملگر Like : از این عملگر برای بررسی تشابه در داده‌های رشته‌ای استفاده می شود. به همراه عملگر like می توان از عملگرهای دیگری نیز استفاده کرد :

```
SELECT * FROM student WHERE FirstName LIKE 'ali %'
```

ب) عملگر - : این عملگر معادل فقط یک کاراکتر است :

```
SELECT * FROM student WHERE FirstName LIKE ' - ali %'
```

ج) عملگر [] : این عملگر برای انتخاب یک کاراکتر از بین چندین کاراکتر بکار می رود :

```
SELECT * FROM student WHERE FirstName LIKE '[ali] %'
```

د) عملگر [^] : این عملگر برای عدم انتخاب از بین چندین کاراکتر است :

```
SELECT * FROM student WHERE FirstName LIKE '[^a-d] %'
```



پارسیابی اطلاعات – بخش Aggregation

برای جمع بندی روی داده‌ها یکسری توابع تحت عنوان تجمعی وجود دارد که لیست آنها را می‌توانید در مسیر زیر بباید :

Programmability ==> Functions ==> System Functions

sum – avg – count – min – max - ...

برخی از این توابع عبارتند از :

نکته: تمام توابع تجمعی به طور اتوماتیک مقادیر null را در نظر نمی‌گیرند به جز یک مورد خاص .

مثال : دستور select ای بنویسید که میانگین سنی دانشجویان رشته فناوری اطلاعات را نمایش دهد :

```
SELECT AVG ( Student . Age ) as 'Average'  
FROM Student INNER JOIN Major  
ON Major . mcode = Student . mcode  
WHERE Major . Mname = N'فناوری اطلاعات'
```

نکته: توابع max و min روی فیلدهای رشته‌ای مقایسه را بر اساس کد اسکی انجام می‌دهند.

نکته: توابع sum و avg فقط بر روی داده‌های عددی قابل استفاده هستند.

نکته: تابع count(*) تنها حالتی است که مقادیر null را محاسبه می‌کند. بنابراین می‌توان از طریق آن تعداد سطرهای جدول را بدست آورد.

مثال : دستور select ای بنویسید که تعداد دانشجویان رشته فناوری اطلاعات را نمایش دهد :

```
SELECT Count(*) as 'Count'  
FROM Student INNER JOIN Major  
ON Major . mcode = Student . mcode  
WHERE Major . Mname = N'فناوری اطلاعات'
```

پاز پایی اطلاعات – پخش Group By

هرگاه بخواهیم گزارشی را به تفکیک فیلد خاصی بدست آوریم از group by استفاده می‌کنیم.
مثال : دستور select ای بنویسید که تعداد دانشجویان دختر و پسر را به تفکیک نمایش دهد :

```
SELECT Gen , COUNT ( * ) as 'Count'  
FROM Student  
GROUP BY Gen
```

نکته (خیلی مهم) : هنگام استفاده از group by ، مقابل select فقط مجاز به استفاده از توابع تجمعی و فیلد/فیلددهای موجود در group by هستیم و هیچ فیلد دیگری را نباید مقابل select نوشت.

نکته : بهتر است فیلدي که دسته بندی بر اساس آن صورت گرفته است را مقابل select بنویسیم.

مثال : دستور select ای بنویسید که میانگین نمرات دانشجویان را به تفکیک نام خانوادگی دانشجو نمایش دهد :

```
SELECT Student . LastName , AVG ( SelectedLesson . Grade )  
FROM Student  
INNER JOIN SelectedLesson ON Student . Stno = SelectedLesson . stno  
GROUP BY Student . LastName
```

پازیابی اطلاعات – بخش Having

بخش where همانند having برای اعمال شرط استفاده می‌شود، با این تفاوت که شرط having بعد از دسته بندی اعمال خواهد شد، در صورتی که شرط where قبل از دسته بندی اعمال می‌شود.

مثال : در این مثال، ابتدا دانشجویان پسر انتخاب می‌شوند (gen = true) ، سپس دسته بندی روی دانشجویان بر اساس نام خانوادگی انجام می‌شود و در نهایت دانشجویان پسری که معدل آنها بیش از ۱۲ باشد نمایش داده می‌شود.

```
SELECT Student . LastName , AVG ( SelectedLesson . Grade )
FROM Student
INNER JOIN SelectedLesson ON Student . Stno = SelectedLesson . Stno
WHERE Student . Gen = True
GROUP BY Student . LastName
HAVING AVG ( SelectedLesson . Grade ) >= 12
```

نکته : هر شرط موجود در where در having قابل استفاده نیست.

نکته : هر شرط موجود در having در where قابل استفاده نیست (توابع تجمعی را نمی‌توان در مقابل where نوشت).

نکته : اگر بتوان شرطی را هم در where و هم در having استفاده کرد، اولویت با where است، زیرا سرعت پرس‌وجو را افزایش می‌دهد.

پازیابی اطلاعات – پخش Order By

برای مرتب سازی داده‌ها از `order by` استفاده می‌شود :

مثال : دستور `select` ای بنویسید که نام خانوادگی و نمره دانشجویان را ابتدا بر اساس نام هاخوادگی (به ترتیب حرف‌الفبا) و سپس بر اساس نمره (نزولی) مرتب کند.

```
SELECT Student . LastName , SelectedLesson . Grade  
FROM Student  
INNER JOIN SelectedLesson ON Student . Stno = SelectedLesson . Stno  
ORDER BY Student . LastName ASC , SelectedLesson . Grade DESC
```

نکته : می‌توان با استفاده از `group by` و `top` گزارشات متنوعی نوشت.

مثال : دستور `select` ای بنویسید که نام و نام خانوادگی و نمره سه نفر از دانشجویانی که در درس پایگاه داده بیشترین نمره را کسب کرده‌اند نمایش دهد.

```
SELECT TOP 3 Student . FirstName , Student . LastName , SelectedLesson . Grade  
FROM Student  
INNER JOIN SelectedLesson ON Student . Stno = SelectedLesson . Stno  
INNER JOIN [Group] ON [Group] . gcode = SelectedLesson . gcode  
INNER JOIN Lesson ON [Group] . lcode = Lesson . lcode  
WHERE Lesson . Lname = N'پایگاه داده'  
ORDER BY SelectedLesson . Grade
```

Union , Intersect , Except

فرض کنید چندین دستور select داریم. خروجی هر select در واقع یک جدول و یا یک مجموعه است :

Select 01 ==> A = { ... }

Select 02 ==> B = { ... }

بنابراین می توان یکسری عملیات مجموعه ها را روی آنها اعمال کرد :

- ۱ - اجتماع (U) : برای این منظور از union استفاده می شود.
- ۲ - اشتراک (\cap) : برای این منظور از intersect استفاده می شود.
- ۳ - تفاضل (-) : برای این منظور از except استفاده می شود.

نکته ۱ : اگر چندین select ترکیب شوند، select order by فقط روی آخرین select ظاهر می شود.

نکته ۲ : اگر چندین select ترکیب شوند، alias فقط روی اولین select ظاهر می شود.

نکته ۳ : select ها باید خروجی های سازگار داشته باشند تا امکان ترکیب آنها وجود داشته باشد. در واقع، تعداد و نوع ستون های خروجی select ها باید یکسان باشد.

پرس و جوی تودر تو (Nested Select)

هنگام نوشتن select های تودر تو ، select داخلی (sub query) را می‌توان به سه صورت مختلف بکار برد :

حالت اول : مقابله select : در این حالت، select داخلی مقابله select قرار می‌گیرد :

```
SELECT ... , ( SELECT ... ) FROM ...
```

نکته : در این حالت حتماً باید select داخلی تک مقداری باشد

نکته : اگر بخواهیم query مفیدی داشته باشیم، بهتر است select داخلی وابسته به select بیرونی باشد تا در تمام سطرهای آن یکسان نباشد (Co-related).

مثال : دستور select ای بنویسید که نام خانوادگی دانشجو را به همراه تعداد واحد اخذ شده هر دانشجو نمایش دهد :

```
SELECT LastName , ( SELECT Count( * ) FROM SelectedLesson WHERE stno = Student . Stno )
FROM Student
```

حالت دوم : مقابله from : در این حالت، select داخلی مقابله from قرار می‌گیرد :

این حالت را در قسمت from شرح داده‌ایم.

```
SELECT FirstName , LastName
FROM ( SELECT TOP 3 * FROM student
      ORDER By Age DESC
      ) as ST
ORDER By LastName
```

حالت سوم : مقابله where : در این حالت، select داخلی مقابله where قرار می‌گیرد : توضیحات در اسلاید بعدی .

پرس و جوی تودر تو (Nested Select) – ادامه

حالت سوم : مقابله **where** : در این حالت، select داخلی مقابله **where** قرار می‌گیرد : توضیحات در اسلاید بعدی .

مثال : دستور select ای بنویسید که نام و نام خانوادگی دانشجویان رشته فناوری اطلاعات را نمایش دهد :

SELECT *FirstName* , *LastName* From *Student*

WHERE *mcode* = (SELECT *mcode* FROM *Major* WHERE *mname* = *N*) 'فناوری اطلاعات'

در این حالت select داخلی و بیرونی از طریق یک فیلد مشترک با هم ارتباط برقرار می‌کنند (*mcode*) .

پرس و جوها (query) در دو نوع هستند :

نوع اول : تک مقداری : برای select های تک مقداری در where می‌توان از عملگرهای ریاضی و مقایسه‌ای استفاده کرد.

نوع دوم : چند مقداری : برای select های چند مقداری نمی‌توان این عملگرها به تنها یک بکار برد و باید آنها را به همراه کلمات all ، any و ya نوشت :

x > All (value1 , value2 , ...) ==> باید از تمام مقادیر مجموعه بزرگتر باشد >

x > any (value1 , value2 , ...) ==> باید از حداقل از یکی از مقادیر مجموعه بزرگتر باشد >

x > some (value1 , value2 , ...) ==> باید از حداقل از یکی از مقادیر مجموعه بزرگتر باشد >

نکته ۱ : عملگر *x = all* معنی دار نیست چون *X* نمی‌تواند با تمام مقادیر مجموعه برابر باشد.

نکته ۲ : به جای عملگر *= any* می‌توان از عملگر *in* استفاده کرد .

مثال : دستور select ای بنویسید که نام درس و تعداد واحد دروسی که استاد احمدی تدریس می‌کند نمایش دهد :

SELECT *Iname* , *unit* From *Lesson*

WHERE *Icode* in (SELECT *Icode* FROM [Group]

WHERE *tcode* in (SELECT *tcode* FROM *Teacher* WHERE *LastName* = *N*)) 'احمدی'

بررسی و جوی نودرتو (Nested Select) – تمرین

تمرین ۱ : دستور `select` ای بنویسید که نام، نام خانوادگی و سن دانشجویان بالای ۲۵ سال را نمایش دهد.

تمرین ۲ : دستور `select` ای بنویسید که نام و نام خانوادگی اساتید مرد را نمایش دهد.

تمرین ۳ : دستور `select` ای بنویسید که نام و نام خانوادگی اساتیدی را که نامشان با حرف «م» شروع می شود را نمایش دهد.

تمرین ۴ : دستور `select` ای بنویسید که نام و نام خانوادگی دانشجویان رشته فناوری اطلاعات را نمایش دهد.

تمرین ۵ : دستور `select` ای بنویسید که نام و نام خانوادگی دانشجویانی که با استاد دادبخش درس دارند را نمایش دهد.

تمرین ۶ : دستور `select` ای بنویسید که نام و نام خانوادگی اساتیدی که درس بانک اطلاعاتی را تدریس می کنند را نمایش دهد.

تمرین ۷ : دستور `select` ای بنویسید که نام درس و تعداد واحد دروسی که استاد دادبخش در نیمسال اول سال ۱۳۹۳ تدریس کرده اند را نمایش دهد.

تمرین ۸ : دستور `select` ای بنویسید که نام و نام خانوادگی و نمره ۳ نفر از دانشجویان رشته فناوری اطلاعات که در نیمسال اول سال ۱۳۹۳ در درس طراحی الگوریتم با استاد دادبخش بیشترین نمره را گرفته اند، نمایش دهد.



آشنایی با اسکریپت نویسی

اسکریپت (script) به مجموعه‌ای از دستورات گفته می‌شود که در یک فایل با پسوند sql ذخیره می‌شوند و ممکن است شامل عملیات مختلفی باشند.

اگر در بخشی از script خطای دستوری یا گرامری (syntax error) رخ دهد، هیچ قسمتی از آن اجرا نخواهد شد.

اگر در بخشی از script خطای در حال اجرا (runtime error) رخ دهد، سایر قسمت‌های قبل از آن اجرا خواهند شد.

می‌توان دستورات یک script را با استفاده از کلمه GO دسته بندی کرد :

هر یک از دسته‌ها را batch می‌گویند.

در واقع، batch مجموعه‌ای از دستورات است در کنار هم قرار می‌گیرند و اجرا می‌شوند.

اگر دسته‌ای خطأ داشته باشد، سایر دسته‌ها بدون هیچ مشکلی اجرا خواهند شد.

(Jump)
پرش

دستورات شرط

تعريف متغير

دستور Case

حلقه تکرار

چاپ پیام

مثال



اسکریپت نویسی – تعریف متغیر

تعریف متغیر :

برای تعریف متغیر از دستور declare به صورت زیر استفاده می شود :

DECLARE `@variable_name data_type` ==> DECLARE `@age int`, DECLARE `@name varchar(20)`

مقداردهی اولیه :

هنگام تعریف متغیر می توان به آن مقدار اولیه نیز نسبت داد :

DECLARE `@variable_name data_type = value` ==> DECLARE `@sum int = 0`

مقدار دهی به متغیر :

برای مقداردهی به متغیر از دستور set به دو صورت استفاده می شود :

SET `@variable_name = value` ==> SET `@name = 'Ali'`

- حالت دوم : مقداردهی با استفاده از select : برای مقداردهی توسط select دو روش وجود دارد :
 - روش اول :

SET `@variable_name = (SELECT ...)` ==>

DECLARE `@count int`

SET `@count = (SELECT Count(*) FROM Student WHERE mcode = '100')`

• روش دوم :

SELECT `@variable_name = ...` ==>

DECLARE `@count int`

SELECT `@count = Count(*) FROM Student WHERE mcode = '100'`)

- با استفاده از روش دوم می توان با یک select چندین متغیر را مقداردهی کرد .



اسکریپت نویسی - دستور شرط

برای انتخاب بین دو یا چند حالت می‌توان از دستور if به صورت زیر استفاده کرد :

<p>حالت سوم :</p> <p>اگر شرط برقرار باشد دستورات داخل begin و end اجرا می‌شوند :</p> <pre>IF condition BEGIN ... END</pre>	<p>حالت اول :</p> <p>اگر شرط برقرار باشد دستور اجرا می‌شود :</p> <pre>IF condition command</pre>
<p>حالت چهارم :</p> <p>اگر برای هر دو حالت برقراری و عدم برقراری شرط چندین دستور را بخواهیم اجرا کنیم دستورات را داخل end و begin قرار می‌دهیم :</p> <pre>IF condition BEGIN ... END ELSE BEGIN ... END</pre>	<p>حالت دوم :</p> <p>اگر شرط برقرار باشد دستور ۱ و اگر شرط برقرار نباشد دستور ۲ اجرا می‌شود :</p> <pre>IF condition Command 1 ELSE Command 2</pre>



اسکریپت نویسی - پرش (Jump)

برای پرش از قسمتی از script به قسمت دیگر آن از goto استفاده می‌شود . این دستور به دو صورت بکار می‌رود :

حالت اول : پرش به سمت پایین :

```
IF condition  
    GOTO L1  
...  
...  
L1 : ...
```

حالت دوم : پرش به سمت بالا :

```
L1 : ...  
...  
...  
IF condition  
    GOTO L1
```

اسکریپت نویسی - چاپ پیام

برای چاپ پیام از دستور print استفاده می‌کنیم :

PRINT 'text'

با دستور پرینت فقط می‌توان خروجی از جنس متن را نمایش داد و اگر بخواهیم خروجی از نوع‌های دیگر را نیز داشته باشیم، ابتدا باید آن را به نوع رشته‌ای (متنی) تبدیل کنیم.
مثال ۱ :

PRINT 'Hi'

مثال ۲ :

```
DECLARE @str varchar(10) = 'Hi'  
PRINT @str
```

نکته ۱ : با استفاده از concat می‌توان دو مقدار از هر نوعی را به هم چسباند.

نکته ۲ : با استفاده از cast می‌توان متغیری از یک نوع را به نوع دیگر تبدیل کرد :

CAST (@variable_name as new_data_type)

مثال ۳ : اسکریپتی بنویسید که تعداد دانشجویان رشته فناوری اطلاعات را ۱۰ درصد افزایش و تعداد دانشجویان رشته سخت افزار را ۵ درصد کاهش دهد. سپس مجموع کل دانشجویان را محاسبه و نمایش دهد :

SET nocount on

UPDATE Major SET stcount = stcount * 1.10 WHERE mname = 'فناوری اطلاعات'

UPDATE Major SET stcount = stcount * 0.95 WHERE mname = 'سخت افزار'

DECLARE @sum int

SET @sum = (SELECT Sum(stcount) FROM Major)

PRINT concat ('Count = ' , @sum) or PRINT 'Count = ' + CAST (@sum as varchar(10))

نکته ۳ : معمولاً با ازای هر update پیامی نمایش داده می‌شود. برای جبوگیری از نمایش این پیام در ابتدای script از دستور set nocount on استفاده می‌کنیم .



اسکریپت نویسی - حلقة تكرار

برای تکرار قسمتی از script بع تعداد دفعات مشخص و یا حتی نامشخص از دستور while به صورت زیر استفاده می‌کنیم :

WHILE <i>condition</i> <i>command</i>	WHILE <i>condition</i> BEGIN ... END
--	---

مثال : جدولی به نام users بسازید که دو فیلد ID و Password داشته باشد. سپس نام کاربری و کلمه عبور ۱۰ کاربر را به طور اتوماتیک تولید کرده و در جدول درج نمایید.

```
SET nocount on
USE University
CREATE TABLE Users
(
    ID varchar(10) not null ,
    Password varchar(10) Not Null
)
DECLARE @i int = 1
WHILE @i <= 10
BEGIN
    INSERT INTO Users
    VALUES ( 'user' + CAST ( @i as varchar(10) ) , @i )
    SET @i = @i + 1
END
SELECT * FROM Users
```

ID	Password
user1	1
user2	2
user3	3
...	...
user10	10

اسکریپت نویسی - دستور Case

برای بررسی چندین حالت مختلف می‌توان از case استفاده کرد که یک دستور نیست، بلکه یک عبارت است و به تنها یعنی ندارد. ساختار case دو حالت دارد :

حالت اول :

CASE *expression*

WHEN *computational_expression11* THEN *computational_expression12*

WHEN *computational_expression11* THEN *computational_expression12*

...

ELSE *computational_expression n*

END

حالت دوم :

CASE

WHEN *conditional_expression11* THEN *computational_expression12*

WHEN *conditional_expression11* THEN *computational_expression12*

...

ELSE *computational_expression n*

END

نکته : روش دوم به دو دلیل بهتر است :

- می‌توان از عبارات شرطی استفاده کرد.
- می‌توان روی فیلدهای مختلف شرط اعمال کرد.

مثال



اسکریپت نویسی - مثال Case

مثال : اسکریپتی بنویسید که تعداد دانشجویان رشته فناوری اطلاعات را ۱۰ درصد افزایش دهد و تعداد دانشجویان رشته سخت افزار را ۵ درصد کاهش دهد.

حالت اول :

```
UPDATE Major  
SET stcount = stcount * CASE mname  
    WHEN 'فناوری اطلاعات' THEN 1.10  
    WHEN 'سخت افزار' THEN 0.95  
END
```

حالت دوم :

```
UPDATE Major  
SET stcount = stcount * CASE  
    WHEN mname = 'فناوری اطلاعات' THEN 1.10  
    WHEN mname = 'سخت افزار' THEN 0.95  
END
```

نکته : روش دوم به دو دلیل بهتر است :

- ۱ - می توان از عبارات شرطی استفاده کرد.
- ۲ - می توان روی فیلدهای مختلف شرطِ اعمال کرد.

مثال اسکریپت نویسی

مثال اول : اسکریپتی بنویسید که با نمایش پیام مناسب مشخص کند دانشجوی رشته فناوری اطلاعات داریم یا خیر.

پاسخ مثال اول

مثال دوم : اسکریپتی بنویسید که قبل از ایجاد جدول users برسی کند که آیا این جدول وجود دارد یا خیر و در صورتی که جدول از قبل موجود باشد پیغام دهد.

پاسخ مثال دوم

پاسخ مثال اول اسکریپت نویسی

روش اول :

```
DECLARE @i int  
SET @i = ( SELECT count(*) FROM Student  
          INNER JOIN Major ON Major . Mcode = Student . Mcode  
          WHERE mname = N'فناوری اطلاعات'  
        )  
If @i = 0  
    PRINT 'دانشجوی فناوری اطلاعات وجود ندارد.'  
ELSE  
    PRINT 'دانشجوی رشته فناوری اطلاعات وجود دارد.'
```

روش دوم :

```
IF EXISTS ( SELECT count(*) FROM Student  
          INNER JOIN Major ON Major . Mcode = Student . Mcode  
          WHERE mname = N'فناوری اطلاعات'  
        )  
    PRINT 'دانشجوی فناوری اطلاعات وجود دارد.'  
ELSE  
    PRINT 'دانشجوی رشته فناوری اطلاعات وجود ندارد.'
```

پاسخ مثال دوم اسکریپت نویسی

```
IF NOT EXISTS ( SELECT * FROM sys . Object WHERE object-id = OBJECT-ID ( 'dbo.users' ) )
BEGIN
    CREATE TABLE Users
    (
        ID varchar(10) not null ,
        Password varchar(10) Not Null
    )
END
ELSE
    PRINT 'جدول مورد نظر وجود دارد.'
IF NOT EXISTS object-id ('dbo.users')
```

شرط فوق را می‌توان به صورت زیر نیز نوشت :

رسیدگی به خطای (Error Handling)

- ممکن است در زمان اجرا، دستورات با خطای مواجه شوند. این نوع خطای runtime error می‌گویند. می‌توان تا حدی از وقوع چنین خطاها را جلوگیری کرد و یا اینکه در صورت بروز خطای پیام مناسب ارائه کرد. برای این منظور از Try catch استفاده می‌کنیم.
- دستوراتی که احتمال بروز خطای آنها وجود دارد در قسمت try و دستوراتی که قرار است در صورت بروز خطای شوند را در قسمت catch می‌نویسیم.
- عملیات داخل try تا قبل از بروز خطای اجرا می‌شوند. به محض بروز خطای کنترل به ابتدای بلوک catch منتقل می‌شود و عملیات داخل آن اجرا نمی‌گردد.
- در قسمت catch یا باید خطای بروز کنیم و یا اینکه پیام مناسب به کاربر ارائه دهیم. در SQL توابعی در مورد خطاها وجود دارد :

Error_Number() ==> کد خطای رخ داده شده را مشخص می‌کند.
Error_Message() ==> در ازای خطای رخ داده شده پیامی ارائه می‌کند.
Error_Line() ==> شماره خطی که خطای رخ داده شده را مشخص می‌کند.
Error_Procedure() ==> روالی که خطای رخ داده شده را مشخص می‌کند.

مثال : می‌خواهیم هنگام ایجاد جدول users در صورتی که جدول موجود باشد پیام مناسبی ارائه شود :

```
BEGIN TRY
CREATE TABLE Users
(
    ID varchar(10) not null ,
    Password varchar(10) Not Null
)
END TRY
BEGIN CATCH
    IF Error_Number() = 2714
        PRINT 'جدول مورد نظر وجود دارد.'
END CATCH
```

```
SELECT * FROM sys . messages
```

نکته ۱ : اگر بخواهیم نتیجه به جای قسمت message در قسمت results نشان داده شود، به جای print از select استفاده می‌کنیم.

نکته ۲ : برای مشاهده لیست خطاها موجود به جدول sys.messages مراجعه کنید :



ترانزکشن (Transaction)

- تراکنش مجموعه‌ای از دستورات است که یا همه اجرا می‌شوند و یا هیچیک اجرا نخواهند شد. این خاصیت را "همه یا هیچ" (All or Nothing) می‌گویند.
 - اگر تمام دستورات تراکنش به درستی اجرا شوند و تراکنش با موفقیت به اتمام برسد، می‌گوییم تراکنش commit شده است.
 - حال اگر به هر دلیلی قسمتی از تراکنش با مشکل مواجه شود، تراکنش ناموفق بوده (Abort) و باید شرایط سیستم به قبل از شروع تراکنش بازگردد (Rollback).
- انواع تراکنش :

۱ - تراکنش ضمنی (Implicit) : هر دستور SQL به طور اتوماتیک یک تراکنش است. مثال زیر را در نظر بگیرید :

```
DELETE FROM Major  
WHERE stcount < 10
```

در این مثال می‌خواهیم تمام رشته‌های تحصیلی که کمتر از ۱۰ دانشجو دارند حذف کنیم. با اجرای این دستور یا باید تمام رشته‌های دارای کمتر از ۱۰ دانشجو حذف شوند و یا اگر مشکلی رخ دهد هیچیک حذف نخواهند شد.

۲ - تراکنش صریح (Explicit) : در این حالت، بیش از یک دستور را می‌خواهیم در قالب تراکنش داشته باشیم. در ساده‌ترین حالت تراکنش به صورت زیر می‌باشد :

```
BEGIN TRAN Major
```

...

```
COMMIT TRAN
```

حالت کامل‌تر بدین صورت است که commit شدن را منوط به شرط یا شرایط خاصی قرار می‌دهیم :

```
BEGIN TRAN
```

...

```
IF Condition
```

```
    Commit
```

```
ELSE
```

```
    Rollback
```

تراکنش تو در تو

مثال تراکنش



ترانزیشن - مثال

مثال : اسکریپتی بنویسید که شماره دانشجویی یک دانشجو را گرفته، اطلاعات دانشجو را حذف کند. سپس اگر تعداد دانشجویان آن رشته تحصیلی کمتر از ۱۰ بود، عملیات لغو شود.

```
DECLARE @c Smallint = 0
SET @c = ( SELECT stcount FROM Major INNER JOIN Student ON Major . mcode = Student . mcode WHERE stno = '1000' )
PRINT 'Old Count = ' + CAST ( @c as varchar(5) )
```

```
BEGIN TRAN
```

```
DECLARE @m char(3)
```

```
SET @m = ( SELECT mcode FROM Student WHERE stno = '1000' )
```

```
DELETE FROM Student WHERE stno = '1000'
```

```
UPDATE Major SET stcount = stcount -1 WHERE mcode = @m
```

```
@c = ( SELECT stcount FROM Major WHERE mcode = @m )
```

```
IF @c < 10
```

```
BEGIN
```

```
ROLLBACK TRAN
```

```
PRINT N'به دلیل کم بودن تعداد دانشجو امکان حذف نمی باشد.'
```

```
END
```

```
ELSE
```

```
BEGIN
```

```
COMMIT TRAN
```

```
PRINT N'عملیات با موفقیت انجام شد.'
```

```
END
```



ترانزکشن تو در تو (Nested Transaction)

در این حالت، یک تراکنش به طور کامل داخل تراکنش دیگر قرار می‌گیرد :

```
BEGIN TRAN  
...  
BEGIN TRAN  
...  
END TRAN  
...  
END TRAN
```

```
BEGIN TRAN  
A  
BEGIN TRAN  
B  
ROLLBACK TRAN  
C  
COMMIT TRAN
```

نکته ۱ : اولین Rollback کل عملیت را لغو می‌کند :

باعث لغو B می‌شود و چون B بخشی از تراکنش بالایی است پس عملیات آن تراکنش نیز لغو می‌شود.

```
BEGIN TRAN  
A  
BEGIN TRAN  
B  
COMMIT TRAN  
C  
ROLLBACK TRAN
```

نکته ۲ : آخرین Commit واقعی است :

باعث لغو A ، C و تراکنش داخلی می‌شود . Commit تراکنش داخلی عمل نخواهد کرد.



(View) دید

یک query ذخیره شده در پایگاه داده را "دید" (view) می‌نامند.

کاربرد "دید":

- تسهیل استفاده
- عملکرد مشابه جدول مجازی است یعنی عملیاتی نظیر درج، ویرایش، حذف و بازیابی داده‌ها را می‌توان روی آن اعمال کرد. البته تغییرات روی جدول اصلی اعمال می‌شود.
- "دید" واسطی برای دسترسی به داده‌ها می‌باشد. با این کار می‌توان جلوی دسترسی مستقیم کاربر به جداول اصلی را لغو کرد و کاربر فقط از طریق "دید" بتواند دسترسی داشته باشد.

عملیات مربوط به "دید":

- : (Create View) ایجاد دید (

```
CREATE VIEW view_name
```

```
AS
```

```
...
```

```
GO
```

- : (Alter View) ویرایش دید (

```
ALTER VIEW view_name
```

```
AS
```

```
...
```

```
GO
```

- : (Drop View) حذف دید (

```
DROP VIEW view_name
```

قواعد تغییر داده

تنظیمات

مثال دید

مثال (View) - دید

مثال ۱ : view ای بنویسید که نام، نام خانوادگی، جنسیت و سن دانشجویان رشته فناوری اطلاعات را نمایش دهد :

USE University

GO

CREATE VIEW *AverageView*

AS

SELECT *FirstName* , *LastName* , *Age* , *Gen* FROM *Student* as *S*

INNER JOIN *Major* as *M* ON *M* . *mcode* = *S* . *Mcode*

WHERE *mname* = 'فناوری اطلاعات'

GO

مثال ۲ : روی view ، *MyView* دیگری بنویسید که میانگین سنی دانشجویان پسر رشته فناوری اطلاعات را نمایس دهد.

USE University

GO

CREATE VIEW *MyView*

AS

SELECT *Average* (*age*) FROM *MyView*

WHERE *Gen* = 1

GO

نکته : view ها را می‌توان به صورت visual نیز مشاهده کرد. برای این منظور روی view کلیک راست کرده و design را اجرا می‌کنیم.



دید (View) – تنظیمات

▪ کنترل ورود اطلاعات با شرط view :

فیلد هایی را که به شرط view ارتباطی ندارند می توان هر مقداری داد، ولی اگر بخواهیم مقدار فیلد مرتبط با شرط view را تغییر دهیم دیگر ممکن است شرط view برآورده نشود و رکورد مورد نظر از view حذف شود.

برای مثال، در MyView لیست دانشجویان فناوری اطلاعات را نمایش می دهیم. حال اگر رشته دانشجو را تغییر دهیم، رکورد آن دانشجو از view حذف می شود.

▪ جلوگیری از تغییرات غیرمجاز در جدول پایه :

اگر در جدول پایه تغییرات به منظور حذف یک فیلد و یا تغییر نام آن باشد view دچار مشکل می شود. برای جلوگیری از این امر باید کاری کنیم که روی view اجازه تغییرات غیرمجاز در جدول پایه را ندهیم. برای این کار قبل از کلمه as از عبارت with schemabinding استفاده می کنیم.

▪ رمزگذاری :

اگر بخواهیم کاربر به source دسترسی نداشته باشد قبل از کلمه as از عبارت with encryption استفاده می کنیم.

دیل (View) - قواعد تغییر داده

- در یک لحظه فقط می‌توان اطلاعات یک جدول را تغییر داد.
- ستون‌های محاسباتی را نمی‌توان تغییر داد و فقط خواندنی هستند.
- ستون‌های گروه بندی شده (group by) را نمی‌توان تغییر داد.
- هنگام تغییرات تمام قواعد و محدودیت‌های جدول کنترل می‌شود و در صورت رعایت تمام محدودیت‌های موجود، اجازه تغییر داده می‌شود.
- با استفاده از تریگر (trigger) می‌توان تمام محدودیت‌های فوق (به جز مورد چهارم) را برطرف کرد.



تابع (Function)

تعريف تابع :

تابع عبارت است از مجموعه‌ای از دستورات که در پایگاه داده ذخیره می‌شوند و شرایط زیر را دارند :

- ۱ - تابع حتماً باید فقط و فقط یک خروجی داشته باشد.
- ۲ - تابع نباید تغییری در پایگاه داده ایجاد کند.
- ۳ - تابع می‌تواند هیچ، یک و یا چند ورودی داشته باشد.

أنواع توابع در sql :

- ۱ - تابع تک مقداری (single valued) : در این حالت خروجی تابع فقط یک مقدار است. هنگام فراخوانی، نام تابع مقابل select قرار می‌گیرد.
- ۲ - تابع جدولی (table valued) : در این حالت خروجی تابع به شکل جدول است. تابع جدولی به دو صورت پیاده سازی می‌شود :
 - الف) inline function
 - ب) multi statement function

هر تابع جدولی مانند view است با این تفاوت که می‌تواند پارامتر ورودی داشته باشد. هنگام فراخوانی، نام تابع مقابل from قرار می‌گیرد.

تنظیمات تابع :

- ۱ - با استفاده از with encryption قبل از as می‌توان کد تابع را از دید کاربر پنهان کرد.
- ۲ - با استفاده از with schemabinding می‌توان از تغییرات غیرمجاز در جداول مرتبط با تابع جلوگیری کرد.
- ۳ - برای مشاهده فهرست وابستگی‌های تابع روی آن کلیک راست کرده و گزینه with dependency را اجرا می‌کنیم.

Single Valued Function

Table Valued Function
(Inline)

Table Valued Function
(Multi Statement)



تابع – Single Valued Function

در این حالت، تابع می‌تواند هیچ، یک و یا چند ورودی داشته باشد ولی فقط یک خروجی دارد.
ایجاد تابع : فرم کلی ایجاد چنین تابعی به صورت زیر است :

```
USE database_name
Go
CREATE FUNCTION function_name
( Input Parameters Definition )
RETURNS output_type
AS
BEGIN
    DECLARE @output_variable output_type
    ...
    RETURN @output_variable / output_value
END
Go
SELECT function_name ( parameters )
```

فراخوانی تابع :

مثال : تابعی بنویسید که جنسیت دانشجو را به عنوان ورودی دریافت کند، سپس تعداد دانشجویان آن جنسیت را بازگرداند.

پاسخ مثال



تابع – Single Valued Function – مثال

مثال : تابعی بنویسید که جنسیت دانشجو را به عنوان ورودی دریافت کند، سپس تعداد دانشجویان آن جنسیت را بازگرداند.

```
USE university
Go
CREATE FUNCTION GenCount
( @G bit )
RETURNS smallint
AS
BEGIN
    DECLARE @count smallint
    SET @count = ( SELECT Count(*) FROM Student WHERE Gen = @G )
    RETURN @count
END
Go
SELECT GenCount ( "True" )
```

فرآخوانی تابع :

تابع – Inline Function

در این حالت، می‌توان یک دستور `Select` نوشت و باید آن را به طور مستقیم مقابل `return` قرار داد :
ایجاد تابع : فرم کلی ایجاد چنین تابعی به صورت زیر است :

```
USE database_name
Go
CREATE FUNCTION function_name
(Input Parameters Definition)
RETURNS TABLE
AS
    RETURN SELECT ...
Go
SELECT * FROM function_name (parameters)
```

فراخوانی تابع :

مثال : تابعی بنویسید که نام و نام خانوادگی دانشجو را به عنوان ورودی دریافت کند، سپس نام درس، تعداد واحد و نمره ۳ درسی که بیشترین نمره را گرفته است نمایش دهد.

پاسخ مثال



مثال – Inline Function – تابع

مثال : تابعی بنویسید که نام و نام خانوادگی دانشجو را به عنوان ورودی دریافت کند، سپس نام درس، تعداد واحد و نمره ۳ درسی که بیشترین نمره را گرفته است نمایش دهد.

USE *university*

Go

CREATE FUNCTION *GradeList*

(@*Fname* nvarchar(30) , @*Lname* nvarchar(30))

RETURNS TABLE

AS

```
RETURN ( SELECT Top 3 Lname , Unit , Grade
        FROM Lesson as L
        INNER JOIN [Group] as G ON G . Lcode = L . Lcode
        INNER JOIN SelectedLesson as SL ON G . Gcode = SL . Gcode
        INNER JOIN Student as S ON S . Stno = SL . Stno
        WHERE FirstName = @Fname and LastName = @Lname
        ORDER BY SL . Grade
    )
```

Go

SELECT * FROM *GradeList* ('ali' , 'ahmadi')

فرآخوانی تابع :

Multi Statement Function – تابع

ایجاد تابع : فرم کلی ایجاد چنین تابعی به صورت زیر است :

```
USE database_name
Go
CREATE FUNCTION function_name
( Input Parameters Definition )
RETURNS @table_name TABLE (
    column_name01 data_type ,
    column_name01 data_type ,
    ...
)
```

```
AS
BEGIN
    دستوراتی که جدول مورد نظر را پر می کند. ... ==>
    در این قسمت چیزی نوشته نمی شود. RETURN ==>
END
Go
SELECT * FROM function_name ( parameters )
```

فراخوانی تابع :

مثال : تابعی بنویسید که نام و نام خانوادگی دانشجو را به عنوان ورودی دریافت کند، سپس نام درس، تعداد واحد، نمره و وضعیت (قبول / رد) را بازگرداند.

پاسخ مثال



مثال – Multi Statement Function – تابع

مثال : تابعی بنویسید که نام و نام خانوادگی دانشجو را به عنوان ورودی دریافت کند، سپس نام درس، تعداد واحد، نمره و وضعیت (قبول / رد) را بازگرداند.

USE *university*

Go

CREATE FUNCTION *GradeList2*

(@*Fname* nvarchar(30) , @*Lname* nvarchar(30))

RETURNS @*ListTable* TABLE (*Name* nvarchar(30) , *Unit* tinyint , *Grade* float , *State* nvarchar(4))

AS

BEGIN

INSERT INTO @*ListTable*

(*Name* , *Unit* , *Grade*)

SELECT *Iname* , *unit* , *grade* FROM *Lesson* as *L*

INNER JOIN [Group] as *G* ON *G* . *Lcode* = *L* . *Lcode*

INNER JOIN *SelectedLesson* as *SL* ON *G* . *Gcode* = *SL* . *Gcode*

INNER JOIN *Student* as *S* ON *S* . *Stno* = *SL* . *Stno*

WHERE *FirstName* = @*Fname* and *Lastname* = @*Lname*

UPDATE @*ListTable* SET *State* = CASE

WHEN *Grade* >= 10 THEN N'قبول'

WHEN *grade* < 10 THEN N'رد'

END

RETURN

END

Go

نام درس	تعداد واحد	نمره	وضعیت
طراحی الگوریتم	۲	۱۸	قبول
پایگاه داده	۲	۱۰	قبول
برنامه نویسی شی گرا	۲	۷	رد

روال یا رویه ذخیره شده (Stored Procedure)

روال، مجموعه‌ای از دستورات ذخیره شده در پایگاه داده می‌باشد و همانند تابع است با این تفاوت که لزومی ندارد خروجی داشته باشد و می‌تواند در پایگاه داده تغییر ایجاد کند.
ایجاد روال : فرم کلی ایجاد روال به صورت زیر است :

```
USE database_name  
CREATE PROC procedure_name  
    Parameters Definition  
AS  
BEGIN  
    ...  
END  
GO
```

انواع پارامتر :

- ۱ - پارامتر ورودی
- ۲ - پارامتر خروجی : بعد از پارامتر خروجی کلمه output قرار می‌گیرد.

فراخوانی روال :

برای فراخوانی روال از دستور Execute (یا exec)

انواع روال :

- ۱ - روال‌های سیستمی (System) : این روال‌ها در پایگاه داده master هستند و به زبان T-sql نوشته شده‌اند. نام آنها با پیشوند sp شروع می‌شود.
- ۲ - روال‌های گسترش یافته (Extended stored procedure) : این روال‌ها در پایگاه داده master و در فولدر C:\Windows\system32\sql\master نوشته شده‌اند. نام آنها با پیشوند xp شروع می‌شود.

- ۳ - روال‌های تعریف شده توسط کاربر (User Defined) : توسط کاربر نوشته می‌شوند.

نکته : می‌توان همانند تابع با استفاده از with encryption قبل از as روال را نیز رمزگذاری کرد.

مثال

روال بازگشتی و تودرتو



روال تودرتو و بازگشتی

: (Nested Stored Procedure)

```
CREATE PROC X
```

...

```
AS
```

```
BEGIN
```

...

```
EXEC Y
```

...

```
END
```

: (Recursive Stored Procedure)

```
CREATE PROC X
```

...

```
AS
```

```
BEGIN
```

...

```
EXEC X
```

...

```
END
```

روال - مثال

مثال ۱ : روالی بنویسید که نام و نام خانوادگی دلنشجویان رشته فناوری اطلاعات را نمایش دهد.

مثال ۲ : روالی بنویسید که نام و نام خانوادگی دلنشجو را دریافت کند، سپس نام درس، تعداد واحد، نمره و وضعیت (قبول / رد) را بازگرداند.

مثال ۳ : روالی بنویسید که جنسیت دانشجو را دریافت کند، سپس دانشجویان آن جنسیت را نمایش دهد. اگر جنسیت وارد نشده بود، همه دانشجویان را نشان دهد.

مثال ۴ : روالی بنویسید که مشخصات یک درس (نام درس، تعداد واحد، کد پیش‌نیاز) از رشته فناوری اطلاعات را گرفته در جدول درس درج کند.

پاسخ مثال ۴

پاسخ مثال ۳

پاسخ مثال ۲

پاسخ مثال ۱



روال - پاسخ مثال اول

مثال ۱ : روالی بنویسید که نام و نام خانوادگی دلنشجویان رشته فناوری اطلاعات را نمایش دهد.

```
USE University
Go
CREATE PROC StudentList
    @mname nvarchar(30)
AS
BEGIN
    SELECT FirstName , LastName
    FROM Student as S
    INNER JOIN Major as M
    ON S . Mcode = M . Mcode
    WHERE mname = @mname
END
GO
EXEC StudentList N'فناوری اطلاعات'
```

فرآخوانی :

روال - پاسخ مثال دوم

مثال ۲ : روالی بنویسید که نام و نام خانوادگی دلنشجو را دریافت کند، سپس نام درس، تعداد واحد، نمره و وضعیت (قبول / رد) را بازگرداند.

USE *University*

GO

CREATE PROC *GradeList*

 @*FName* nvarchar(30) , @*LName* nvarchar(30)

AS

BEGIN

 SELECT *Iname* , *unit* , *grade* , *state* = CASE

 WHEN *Grade* >= 10 THEN N'قبول'

 WHEN *grade* < 10 THEN N'رد'

 END

 FROM *Lesson* as *L*

 INNER JOIN [Group] as *G* ON *G* . *Lcode* = *L* . *Lcode*

 INNER JOIN *SelectedLesson* as *SL* ON *G* . *Gcode* = *SL* . *Gcode*

 INNER JOIN *Student* as *S* ON *S* . *Stno* = *SL* . *Stno*

 WHERE *FirstName* = @*Fname* and *LastName* = @*Lname*

END

GO

EXEC *GradeList* N'احمدی' , N'علی'

: فراخوانی

روال - پاسخ مثال سوم

مثال ۳ : روالي بنويسيد که جنسیت دانشجو را دریافت کند، سپس دانشجویان آن جنسیت را نمایش دهد. اگر جنسیت وارد نشده بود، همه دانشجویان را نشان دهد.

USE *University*

Go

CREATE PROC *StudentList*

 @gen bit = Null

AS

BEGIN

 IF @gen IS NULL

 SELECT * FROM *Student*

 ELSE

 SELECT * FROM *Student* WHERE *Gen* = @gen

END

GO

می توان بدنه روال را به صورت زیر نیز نوشت :

SELECT * FROM *Student* WHERE *Gen* = @gen and @gen IS NULL

فراخوانی با پارامتر :

EXEC *StudentList* 'True'

فراخوانی بدون پارامتر :

EXEC *StudentList*

روالی - پاسخ مثال چهارم

مثال ۴ : روالی بنویسید که مشخصات یک درس (کد درس، نام درس و تعداد واحد) از رشته فناوری اطلاعات را گرفته در جدول درس درج کند.

USE *University*

GO

CREATE PROC *InsertLesson*

 @*lcode* char(6) , @*lname* nvarchar(30) , @*unit* tinyint

AS

BEGIN

 INSERT INTO *Lesson*

 (*lcode* , *lname* , *unit*)

 VALUES (@*lcode* , @*lname* , @*unit*)

END

GO

EXEC *InsertLesson* '1000' , N' طراحی الگوریتم' , 2

فرآخوانی :

(Trigger) ټریگر

تریگر حالت خاصی از stored procedure است با این تفاوت که توسط کاربر قابل فراخوانی نیست و به طور اتوماتیک در زمان رویداد خاصی فراخوانی می‌شود.

انواع تریگر :

(delete , update , insert) DML – ۱

(drop , alter , create) DDL – ۲

این دو نوع تریگر از نظر رویداد، شی مورد نظر، زمان رخ دادن و تعداد با هم تفاوت دارند.

تعداد	زمان اجرا	شی مورد نظر	رویداد	
n	After Instead of	Table View	Insert Update Delete	DML
1	After	Database Server	Create Alter Drop	DDL

After : در این حالت، اتفاق رخ می‌دهد سپس روال اجرا می‌شود.

Instead of : در این حالت، اتفاق رخ نمی‌دهد و روال به جای آن اجرا می‌شود.

تعداد تریگرهای DML نامحدود است ولی تریگر از نوع DDL فقط یک مورد می‌تواند باشد.

مثال

Trigger

DML



DML – (Trigger) تریگر

فرم کلی تریگر DML به صورت زیر است :

```
USE database_name
CREATE TRIGGER trigger_name
ON table_name / view_name
AFTER / INSTEAD OF event_name
AS
BEGIN
...
END
```

کاربردهای تریگر DML :

- جلوگیری از تغییرات یا داده‌های غیرمجاز : چنین کنترل‌هایی معمولاً روی insert و update انجام می‌شود. اعتبارسنجی (validation) های ساده را می‌توان با check constraint انجام داد. از تریگر برای موارد پیچیده استفاده می‌شود.
- ثبت عملیات کاربران
- پیاده سازی قواعد جامعیت پیچیده
- ایجاد امکان اعمال تغییرات در view

DDL – (Trigger) تریگر

این نوع تریگرها روی پایگاه داده یا server نوشته می‌شوند و هنگام رویدادهایی نظیر create ، alter و drop فراخوانی می‌شوند.
فرم کلی تریگر DDL به صورت زیر است :

```
USE database_name
CREATE TRIGGER trigger_name
ON all server / database
AFTER event_name
AS
BEGIN
    ...
END
```

مثال – (Trigger) ۳

مثال ۱ : تریگری بنویسید که اجازه تغییر نمره را ندهد .

```
USE University
CREATE TRIGGER tr_ChangeGrade
ON SelectedLesson
AFTER Update
AS
BEGIN
    IF update ( Grade )
        ROLLBACK
END
```

مثال ۲ : تریگری بنویسید که محدوده تعداد واحد را از یک تا چهار در نظر بگیرد.

```
USE University
CREATE TRIGGER tr_UnitNumber
ON Lesson
AFTER Insert , Update
AS
BEGIN
    IF update ( Unit )
        IF NOT EXISTS ( SELECT Unit FROM Inserted WHERE Unit <= 1 and Unit >= 4 )
            ROLLBACK
        PRINT N' 'محدوده تعداد واحد باید بین یک تا چهار باشد.'
END
```



پایان فصل چهارم

مهدی دادبخش

mahdi.dadbakhsh@sharif.edu

۱۴۰۱ - ۱۴۰۲